République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abou BakrBelkaid – Tlemcen

Faculté de Technologie

Département de Génie Electrique et Electronique

Mémoire de Fin d'Etudes

Pour l'obtention du diplôme de Master en Génie industriel

Spécialité : Productique

Thème :

Une métaheuristique basée sur le comportement des lucioles pour la résolution d'un problème d'ordonnancement dans un atelier flow shop

Préparé par :

MAMOUNI Adel

OULD MOHAMED Belkacem

Soutenu en 28 Juin 2018 devant le jury composé de :

Président : GUEZZEN Amine	MCB – Université de Tlemcen
Examinateur : BESSENOUCI Hakim Nadhir	MAA – Université de Tlemcen
Examinateur : HADRI Abdelkader	MAA – Université de Tlemcen
Encadreur : HOUBAD Yamina	MAA – Université de Tlemcen
Co-encadreur : BOUMEDIENE Fatima Zohra	Doctorante – Université de Tlemcen

Année Universitaire : 2017/2018

Table des matières

LISTE DES FIGURES

LISTE DES ALGORITHMES

LISTE DES TABLEAUX

Dédicace

Nous dédions ce travail à

Nos mères, sources de tendresse et d'amours pour leurs soutiens tout le long de notre vie scolaire.

Nos pères, qui nous ont toujours soutenus et qui ont fait tout possible pour nous aider.

Nos frères et nos sœurs, que nous aimons beaucoup.

Notre grande famille.

Nos cher amis, et enseignants.

Tous ce qui ont collaboré de près ou de loin à l'élaboration de ce travail.

Que dieu leur accorde santé et prospérité

Remerciements

Tout d’abord, nous remercions le Dieu, notre créateur de nous avoir donné les forces, la volonté et le courage afin d’accomplir ce travail.

Nous adressons le grand remerciement à notre Encadreur mademoiselle **HOUBAD Yamina** et notre Co-encadreur mademoiselle **BOUMADIENE Fatima Zohra** qui ont proposé le thème de ce mémoire, pour leurs conseils et dirigés du début à la fin de ce travail. Nous tenons à leur faire part de toute notre gratitude pour nous avoir accordé tant de confiance.

Nous souhaitons remercier monsieur **GUEZZEN Amine**, qui nous a fait l’honneur d’exercer les fonctions de président du jury et d’examinateur de thèse pour le temps qu’il a consacré à étudier notre travail et pour sa sympathie.

Nos remerciements s’adressent également aux examinateurs de notre mémoire du projet de fin d’étude monsieur **BESSENOUCI Hakim Nadhir** et monsieur HADRI Abdelkader pour avoir participer au jury de notre travail et avoir consacré de leur temps pour l’examiner. Nous leur exprimons toute notre reconnaissance pour l’intérêt porté à ce travail.

Finalement, nous tenons à exprimer notre profonde gratitude à nos familles qui nous ont toujours soutenues et à tout ce qui participé à réaliser ce mémoire. Ainsi que l’ensemble des enseignants qui ont contribué à notre formation.

Résumé

L’objectif de ce notre travail est l’adaptation de deux métaheuristiques à base de population de solutions, l’algorithme génétique et l’algorithme de luciole pour la résolution d’un problème d’ordonnancement Flow Shop. Le choix de ces deux techniques est dû au fait qu’elles ont une certaine ressemblance dans leurs principes de recherche.

Les résultats de simulation ont montré que l’algorithme de luciole dépasse l’algorithme génétique en termes de qualité des solutions obtenues pour différentes classes de problèmes.

Mots clés

Ordonnancement, Algorithme génétique, Algorithme de luciole, Flow Shop.

Abstract

The purpose of our work is the adaptation of two population based metaheuristics, the genetic algorithm and the firefly algorithm to solve a Flow Shop Scheduling problem. The choice of the two techniques is due the fact that they have some resemblance in their search technique principles

Computational results have shown that the firefly algorithm technique outperform the genetic algorithm in terms of the solutions quality obtained for different classes of problems.

Key words

Scheduling, Genetic algorithm, Firefly algorithm, Flow Shop.

ملخص

الهدف من هذا المشروع هو عمل محاكاة لتطبيق فوقيتي استدلال متعددة الحلول وهما خوارزمية الجينات وخوارزمية اليراع من اجل حل إشكالية الترتيبات وحيدة المسار. اختيار هاتين التقنيتين يعود لكونهما متشابهتين في مبادئ تقنية البحث.

نتائج المحاكاة أظهرت أن تقنية خوارزمية اليراع تخطت خوارزمية الوراثة من حيث نوعية الحلول المحصل عليها لعدة طبقات من الإشكاليات.

الكلمات المفتاحية

الترتيبات، خوارزمية الوراثة، خوارزمية اليراع، أحادي المسار.

INTRODUCTION GENERALE

Dans les dernières années, le développement rapide de la technologie de l'information avec l'intensification de la tendance de la mondialisation économique, fait que les systèmes de production doivent faire face de plus en plus à la compétition au niveau du délai d'achèvement des produits. Les décisions des entreprises ont un impact significatif sur la fabrication, le coût du produit, le délai de livraison, la qualité… etc., pour les améliorer, elles doivent optimiser la planification et l'ordonnancement de la production. Lorsque la formulation d'un problème d'optimisation conduit à un problème avec une grande complexité on fait appel aux méthodes dites approchées vue que ces techniques offrent la possibilité de trouver des solutions, généralement, de bonne qualité en un temps raisonnable.

Dans notre travail une étude de recherche qui a pour objectif la résolution d'un problème d'ordonnancement dans un atelier de type flow shop avec l'utilisation d'une métaheuristique relativement récente l'algorithme de luciole. Ce mémoire s'articule autour de trois chapitres :

Le premier chapitre décrit les notions et définitions des problèmes d'ordonnancement dans les systèmes de production et leur formulation.

Le deuxième chapitre donne une classification des méthodes de résolution du problème d'ordonnancement suivi par une partie sur les métaheuristiques et leur classification ensuite nous présentons différents algorithmes (Le recuit simulé, La recherche tabou, Les colonies de fourmis…) leurs définitions, leurs phénomènes d'inspiration et leurs paramètres.

Le troisième chapitre représente notre problème étudié suivi par l'adaptation de l'algorithme de luciole et les résultats de simulations ainsi que la comparaison avec l'algorithme génétique.

Chapitre I :

Formulation des problèmes d'ordonnancement

1.1 Introduction

La résolution des problèmes d'ordonnancement consiste à optimiser une fonction objectif en déterminant la séquence suivant laquelle un ensemble des tâches doit être exécuté sur un ensemble des ressources et soumis à des contraintes, ainsi qu'à déterminer les instants de début et de fin d'exécution des différentes tâches sur chacune des ressources.

Les problèmes d'ordonnancement existent dans de nombreux secteurs d'activités comme la gestion de la production, dans l'industrie manufacturière ou encore les systèmes informatiques, où les tâches représentent les programmes et les ressources sont les processeurs ou la mémoire, la conception des emplois des temps, etc.

Parmi ces nombreux types de problèmes d'ordonnancement, nous nous sommes intéressés dans le cadre de ce mémoire aux problèmes d'ateliers. Dans ces problèmes, l'ensemble des opérations (tâches) qui doivent être exécutées sur les différentes machines (ressources) constitue un travail appelé job.

En fonction de l'ordre de passage des opérations sur les machines, on distingue trois classes de problèmes, à savoir :

- Le problème Flow-Shop : Tous les jobs suivent le même ordre de passage sur chacune des machines.
- Le problème Job-Shop : Chaque job suit une gamme qui lui est propre et chacun d'entre eux peut exécuter plusieurs opérations sur une même machine.
- Le problème Open-Shop : L'ordre d'exécution des opérations qui composent un job n'est pas fixé à l'avance et peut donc être différent d'une solution proposée du problème à une autre.

1.2 Généralités sur l'ordonnancement

Ordonnancer le fonctionnement d'un system industriel de production consiste à gérer l'allocation des ressources au cours du temps, tout en optimisant au mieux un ensemble de critère. C'est aussi programmer l'exécution d'une réalisation en attribuant des ressources aux taches et en fixant leurs dates d'exécution **[1].**

Ordonnancer peut également consister à programmer l'exécution des opérations en leur allouant les ressources requises et en fixant leurs dates de début de fabrication. D'une manière plus simple, un problème d'ordonnancement consiste à affecter des tâches à des moyens de fabrication au cours du temps pour effectuer un ensemble de travaux de manière à optimiser certain critère, tout en respectant les contraintes techniques de fabrication **[2]**. L'ordonnancement se déroule en trois étapes qui sont :

La planification, qui consiste à déterminer les différentes opérations à réaliser les dates correspondantes, et les moyens matériels et humains à y affecter.

L'exécution, qui vise à mettre en œuvre les différentes opérations définies dans la phase de planification.

Le contrôle, qui effectue une comparaison entre la planification et l'exécution soit au niveau des coûts, soit au niveau des dates de réalisation.

Ainsi, le résultat d'un ordonnancement est un calendrier précis de tâches à réaliser qui se décompose en trois caractéristiques importantes :

- L'affectation, qui attribue les ressources nécessaires aux tâches.
- Le séquencement, qui indique l'ordre de passage des tâches sur les différentes ressources.
- Le datage, qui indique le temps de début et de fin d'exécution des tâches sur les différentes ressources.

La solution d'un problème d'ordonnancement générale doit répondre à deux question :

- Quand ?
- Avec quels moyens ?

Une solution répandant à ces questions est appelée ordonnancement. Une méthode permettant de construire un ordonnancement est appelée algorithme ou méthode de résolution. Un ordonnancement dit réalisable est un ordonnancement qui respecte toutes les contraintes du problème. Nous utilisons le terme« ordonnancement » pour simplifier, pour représenter un ordonnancement réalisable.

1.3 Les données d'un problème d'ordonnancement

L'ordonnancement de production a fait durant plusieurs années l'objet de recherches très approfondies pour les nombreux problèmes identifiés dont la résolution repose sur une politique d'ordonnancement qui soit la plus optimale possible, ce qui a permis de mettre au point de nombreuses résolutions**.**

Etablir un ordonnancement revient donc à coordonner l'exécution de toutes les tâches, en utilisant au mieux les ressources disponibles **[3]**. En d'autres termes, il s'agit de : « Déterminer ce qui va être fait, quand, où et avec quels moyens, étant donnée un ensemble de tâche accomplir, le problème d'ordonnancement consiste à déterminer quelles tâches doivent être exécutées et à assigner des dates et des ressources à ces tâches de façon à ce que les tâches soient, dans la mesure possible, accomplies en temps utile, au moindre coût et dans les meilleures condition »**[4].**

D'après cette explication, on remarque que dans un problème d'ordonnancement quatre éléments fondamentaux interviennent : les tâches, les ressources, les contraintes et les objectifs.

1.3.1 Les tâches

Une tâche est un travail mobilisant des ressources et réalisant un progrès significatif dans l'état d'avancement du projet compte rendu du niveau de détail retenu dans l'analyse du problème. **[5].**

On note générale i= {1, 2, …, n} l'ensemble des tâches et p_i la durée de la tâche i si cette durée ne dépend pas des ressources qui lui sont allouées **[6]**.

En plus de sa durée, une tâche a d'autres caractéristiques :

- r_i : date de disponibilité, date avant laquelle la tâche i ne peut pas commencer.
- di : date échue, une tâche i doit être achevée avant sa date échue.
- t_i : date réelle de début de la tâche i, date sera qui déterminée uniquement pendant l'ordonnancement.
- c_i : date de fin réelle de la tache i, date qui sera elle aussi calculée uniquement pendant l'ordonnancement.

Les tâches sont souvent liées entre elles par des relations d'antériorité. Si ce n'est pas le cas, on dit qu'elles sont indépendantes. La contrainte d'antériorité la plus générale entre deux

tâches i et j est appelée contrainte potentielle.

Quand la tâche j commence à la fin de la tâche i, dans ce cas on dit qu'il y a une succession simple.

$$\forall\, i = \{1,2, \dots \dots \dots \dots, n\}, ri \leq ci \leq di$$

1.3.2 Les ressources

Une ressource est un moyen technique ou humain utilisé pour permettre la réalisation des tâches. Ce moyen technique est donc nécessaire et indispensable pour le bon fonctionnement du cycle de fabrication **[5].** Plusieurs types de ressources sont à distinguer :

1.3.2.1 Les ressources renouvelables

Une ressource est renouvelable si après avoir été allouée à une ou plusieurs tâches, elle est à nouveau disponible en même quantité (les ressources humaines, les machines, l'équipement) ; la quantité de ressources utilisable à chaque instant est limitée. Parmi les ressources renouvelables, nous distinguons les ressources disjonctives qui ne peuvent exécuter qu'une opération à la fois ; et les ressources cumulatives qui elles peuvent exécuter simultanément un nombre limité d'opérations.

1.3.2.2 Les ressources consommables

Une ressource est consommable ou non renouvelable si après avoir été allouée à une ou plusieurs tâches, ne sont plus disponibles, et sont donc épuisées (matières premières, budget, …), la consommation globale au cours du temps est limitée. De plus, lorsqu'une tâche n'a besoin que d'une seule ressource pour pouvoir être exécutée, on dira alors que c'est un problème d'ordonnancement mono-ressource, et dans le cas contraire où l'exécution d'une tâche nécessite plusieurs ressources c'est un problème d'ordonnancement multi-ressources.

1.3.3 Les contraintes

Les contraintes représentent les conditions à respecter lors de construction de l'ordonnancement pour qu'il soit réalisable. Elles rendent les problèmes d'ordonnancement plus difficiles car il faut les respecter lors de la résolution de ces problèmes **[2]**. Ces contraintes peuvent être classées en deux types :

Le premier type de contraintes constituent des contraintes liées directement au système de production et à ses performances telles que :

- Les dates de disponible des machines et des moyens de transport.
- Les capacités des machines.
- Les séquences des actions à effectuer ou les gammes des produits.

Le deuxième type de contraintes regroupe celles qui sont imposées extérieurement. Elles sont indépendantes du système de production :

- Les dates de fin de fabrication au plus tard de produit imposées généralement par les commandes.
- Les priorités de quelques commandes et de quelques clients.
- Les retards possibles accordés pour certains produits.
- Les contraintes de capacités, caractérisés par un ensemble de conflits pour l'utilisation des ressources
- Les contraintes de gamme, caractérisant l'ordre d'exécution des taches selon la gamme
- Contraintes cumulatives où il y a partage de plusieurs ressources par plusieurs taches.
- Les contraintes disjonctives, une seule ressource est partagée par plusieurs taches à la fois.

1.3.4 Les objectifs

Lors de la résolution d'un problème d'ordonnancement, nous pouvons choisir entre deux principaux types de stratégies, en se concentrant respectivement sur l'optimalité des solutions (par rapport à un ou plusieurs critères), ou sur leur faisabilité (en fonction des contraintes). L'approche fondée sur l'optimisation suppose que les solutions candidates à un problème peuvent être vérifiées, selon un ou plusieurs critères d'évaluation qui expriment leurs qualités **[5]**. Généralement on distingue plusieurs classes d'objectifs concernent un ordonnancement donné :

- Liés au temps : Lorsqu'il s'agit de la minimisation du temps total d'exécution, du temps de cycle, des durées totales de réglage ou des retards par rapport aux dates de livraison…
- Liés aux ressources : Comme par exemple pour la maximisation du taux

d'utilisation d'une ressource ou la minimisation du nombre de ressources à employer…

- Liés au coût : Ces critères permettent de minimiser les coûts de lancement, de production, de stockage, de transport, du retard, de non occupation des machines…

1.4 Problèmes d'ordonnancement d'ateliers

Un problème d'ordonnancement d'atelier consiste à déterminer la séquence de passage d'un certain nombre d'opérations à exécuter sur différentes machines **[7]**.

Une classification très répandue des ateliers, du point de vue ordonnancement, est basée sur les différentes configurations des machines. Les modèles les plus connus sont ceux problèmes d'une machine unique, problèmes des machines parallèles, problèmes d'un atelier à cheminement unique (Flow shop), problèmes d'un atelier à cheminement multiple (Job shop) et problèmes d'atelier multi-machines à cheminements libres (Open shop) **[8], [9], [10]**.

1.4.1 Le type d'une machine unique

Dans ce cas, l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine.

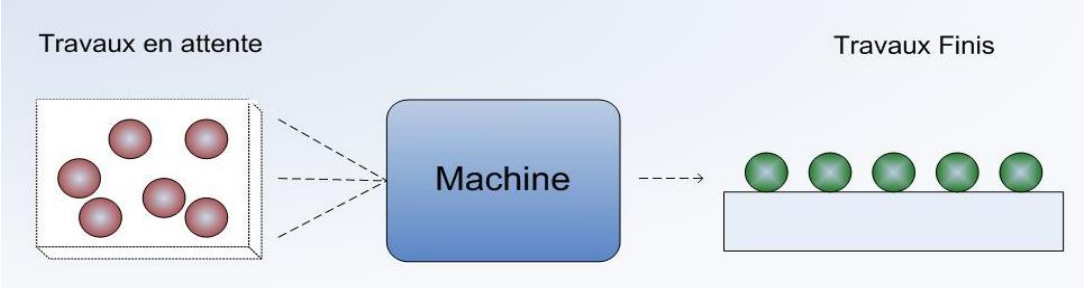


Figure 1.1 : Exemple d'une seule machine

1.4.2 Le type des machines parallèles

Il représente une généralisation du problème d'ordonnancement à une machine, les travaux se composent d'une seule opération et chaque opération dispose d'un ensemble de machines parallèles, mais n'en nécessite qu'une pour pouvoir être réalisée. L'ordonnancement s'effectue en deux phases : la première consiste à affecter les opérations aux machines et la deuxième détermine la séquence de réalisation sur chaque machine.

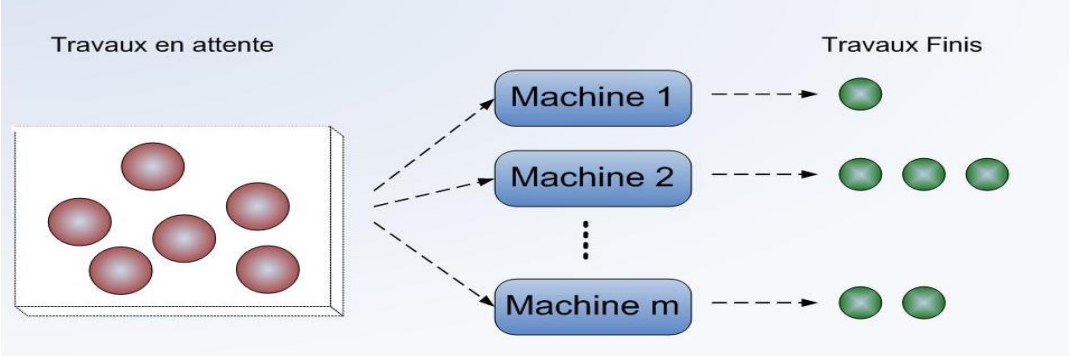


Figure 1.2 : Exemple des machines parallèles

1.4.3 Le type flow shop

Dans ce cas, une ligne de fabrication est composée de plusieurs machines disposées en série ; une tâche est alors constituée de plusieurs opérations qui passent par différentes machines, de manière linéaire, c'est-à-dire dans le même ordre

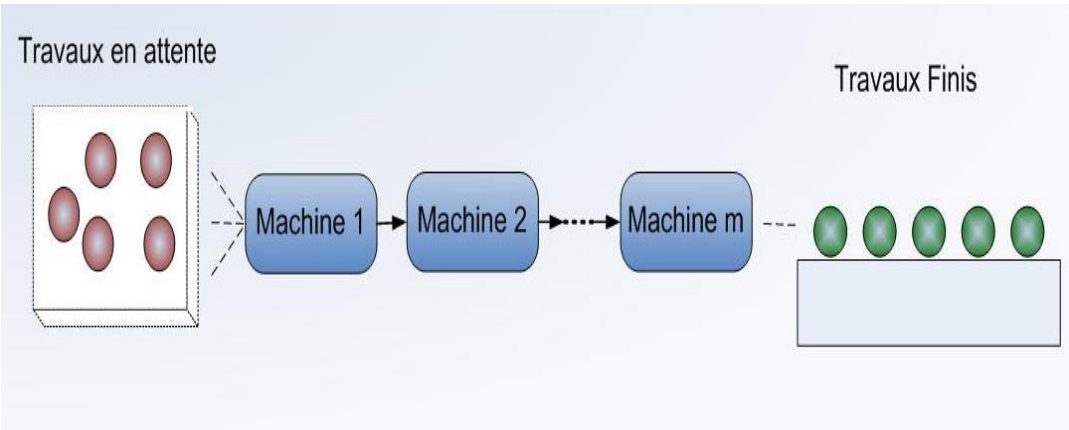


Figure 1.3 : Exemple d'un atelier flow shop

Dans le domaine flow shop, on distingue plusieurs types de flow shop :

- **Flow shop hybride :** c'est un atelier flow shop dans lequel chaque machine remplacée par un étage composé plusieurs des machines qui ne sont pas forcément identique et déposées en parallèles. Chaque travail qui visite successivement chaque étage et ne doit passer que par une seule machine par étage.
- **Flow shop pur** : tous les temps opératoires sont positifs
- **Flow shop généralisé** : les temps opératoires peuvent être nuls si une tache ne doit pas subir un traitement sur une machine particulière.
- **Flow shop de permutation** : toutes les taches sont disponible à l'instant 0, les taches s'exécutent dans l'ordre défini à l'instant 0, le dépassement n'est pas autorisé.

1.4.4 Le type job shop

Contrairement aux problèmes d'ordonnancement Flow shop, les tâches ne s'exécutent pas sur toutes les machines selon un ordre bien déterminé, chaque tâche peut emprunter son propre chemin.

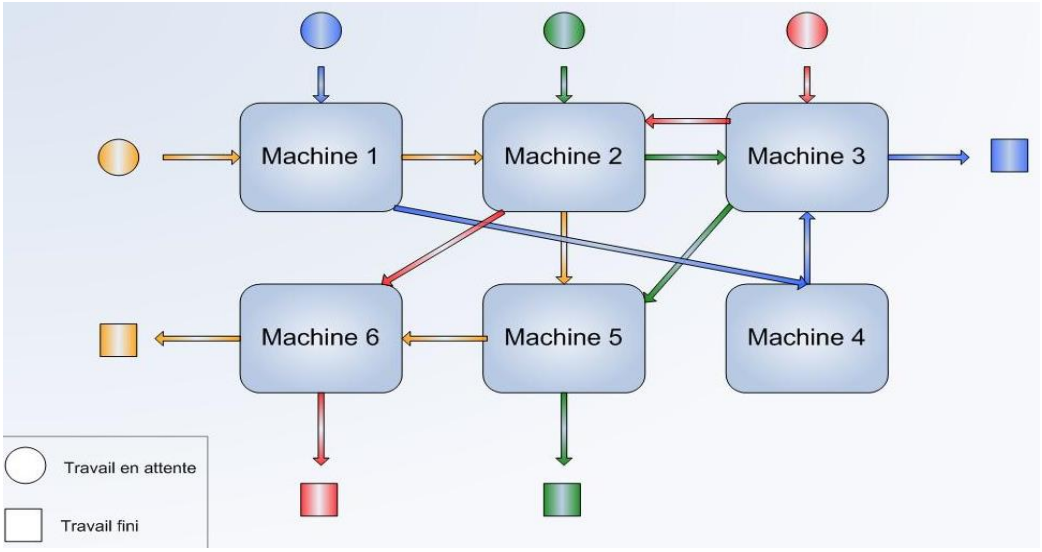


Figure 1.4 : Exemple d'un atelier job shop

1.4.5 Le type open shop

Dans ce type de problème, les opérations nécessaires à la réalisation de chaque tâche peuvent être exécutées dans un ordre quelconque. Autrement dit, ce cas se présente lorsque chaque produit à fabriquer doit subir une séquence d'opérations, mais dans un ordre totalement libre c'est-à-dire le développeur de l'ordonnancement est fixer le routage de chaque tache.

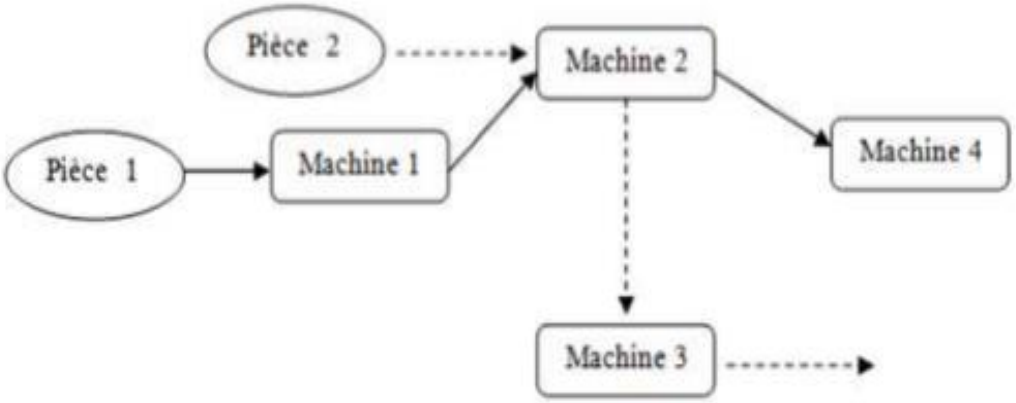


Figure 1.5 : Exemple d'un atelier open shop

1.5 Les critères d'optimisation

Dans la résolution d'un problème ordonnancement, on peut choisir entre deux grands types objectifs, visant respectivement à l'optimalité des solutions ces ta dire trouver la meilleure solution, ou le plus simplement à leur admissibilité (trouver une solution).

Les critères que doit satisfaire un problème d'ordonnancement sont variés avec le temps ou variés de problème à autre problème. Généralement il existe une manière, on peut distinguer trois catégories importantes **[11]** :

- Les critères liés aux dates de fin et dates de livraison.
- Les critères liés aux volumes des encours.
- Les critères liés à l'utilisation des ressources.

1.6 La complexité et la théorie de la complexité

Afin de mesurer la difficulté d'un problème donné et la comparer avec celles d'autres problèmes pour pouvoir dire qu'un tel problème est plus facile à résoudre que l'autre, nous pouvons calculer la complexité algorithmique de chacun d'entre eux. La complexité d'un problème donné est discutée sur deux côtés : coté temporel (complexité temporelle) et coté spatial (complexité spatiale). La complexité temporelle consiste à évaluer le temps de calcul nécessaire pour résoudre un problème donné. Tandis que la complexité spatiale permet d'estimer les besoins en mémoire (l'espace mémoire requis) pour la résolution d'un

problème donné. La complexité d'un problème donné est estimée en fonction du nombre d'instructions permettant d'aboutir à la solution du problème posé. Elle est influencée par la taille du problème en question. En effet, elle exprime un rapport entre la taille du problème, le temps de calcul nécessaire et l'espace mémoire requis **[12], [13].**

La théorie de la complexité s'intéresse à l'évaluation de la difficulté des problèmes via l'étude de la complexité de solutions algorithmiques proposées. Elle associe une fonction de complexité à chaque algorithme résolvant un problème donné et mesure les ressources nécessaires pour la résolution d'un problème posé. En effet, elle classe l'ensemble des problèmes selon deux critères qui sont le temps de calcul nécessaire et l'espace mémoire requis.

Bien que la théorie de la complexité se concentre sur des problèmes de décision, elle peut être étendue aux problèmes d'optimisations. Elle classe les problèmes selon leur complexité en deux classes principales : la classe P (Polynomial time) et la classe NP (Non deterministic Polynomial time). En outre, elle partage les problèmes de la classe NP en deux sous classes : NP-Complet et NP-Difficile **[14].**

1.6.1 Les problèmes de la classe P

Les problèmes appartenant à la classe P sont des problèmes pouvant être résolus par une machine de Turing déterministe (Une machine abstraite représentant un modèle abstrait du fonctionnement de l'ordinateur et de sa mémoire. Elle a été proposée par le mathématicien Alan Mathisan Turing en 1936 en vue de donner une définition précise au concept d'algorithme.) en temps de calcul polynomial par rapport à la taille de l'instance du problème à traiter. Ils sont souvent faciles à résoudre par des algorithmes efficaces dont le nombre d'instructions nécessaire pour la résolution d'un problème donné est borné par une fonction polynomiale par rapport à la taille du problème **[15].**

1.6.2 Les problèmes de la classe NP

La classe des problèmes NP regroupe l'ensemble de problèmes qui peuvent être résolus avec une machine de Turing non-déterministe (Le terme non- déterministe désigne un pouvoir qu'on incorpore à un algorithme pour qu'il puisse aboutir à la bonne solution) et admettent un algorithme polynomial pour tester la validité d'une solution du problème traité c'est à dire il est possible de vérifier que la solution proposée est correcte en un temps polynomial par

rapport à la taille du problème. Le problème du sac à dos, le problème du voyageur de commerce, les problèmes d'ordonnancement sont des exemples de problèmes appartenant à la classe NP **[16]**.

1.6.2.1 Les problèmes de la classe NP-Complets

La classe NP-Complet regroupe les problèmes de décision pour lesquels il n'existe pas d'algorithme permettant leur résolution en un temps polynomial. Selon Garey et Johnson, un problème X est un problème NP-Complet s'il appartient à la classe NP, et si quel que soit le problème X'qui appartient aussi à la classe NP, on peut le réduire au problème X en un temps polynomial**[17]**.

1.6.2.2 Les problèmes de la classe NP-Difficiles

La classe de problèmes NP-Difficiles englobe les problèmes de décision et les problèmes d'optimisation. Les problèmes NP-Difficiles sont aussi difficiles que les problèmes NP-Complets. Si un problème de décision associé à un problème d'optimisation P est NP-Complet alors P est un NP-Difficile [Charon et al, 1996]. Par conséquent, afin de prouver qu'un problème d'optimisation est NP-Difficile, il suffit de montrer que le problème de décision associé à P est NP-Complet [Layeb, 2010]. Il est à noter que jusqu'à maintenant, aucun algorithme polynomial n'est connu pour résoudre ce type de problèmes (i.e. NP-Difficiles) **[18].**

1.7 Méthodes de résolution

Les problèmes d'ordonnancement, sont généralement similaires à des problèmes d'optimisation combinatoire, ils peuvent être résolus en utilisant deux méthodes différentes, la première méthode consiste à utiliser des algorithmes exacts, elle est appliquée à des problèmes de petites tailles et conduit à des solutions optimales, et la deuxième méthode est caractérisée par l'utilisation de méthodes approches, elle est appliquée à des problèmes de grands tailleset les problèmes plus complexes, conduit à solution quasi optimale**[19]**.

1.7.1 Méthodes exactes

Les méthodes exactes sont des techniques de résolution, l'intérêt cette des méthodes réside dans le fait qu'elles assurent l'obtention de la solution optimale du problème traité, les méthodes exactes sont très connues par le fait qu'elles nécessitent un coût de recherche souvent prohibitif en termes de ressources requises, et le temps de recherche (l'espace

mémoire) nécessaire pour l'obtention de la solution optimale par une méthode exacte sont souvent grands**[2]**.

Les méthodes exactes sont multiples et variées telles que la programmation linéaire, la programmation dynamique et Branch and Bound …etc.

1.7.1.1 La programmation linéaire

La programmation linéaire est une approche générique basée sur la modélisation mathématique des problèmes d'optimisation combinatoires, où la fonction objective, les contraintes et les variables de décisions sont linéaires.

1.7.1.2 La programmation dynamique

La programmation dynamique est un paradigme de programmation, l'efficacité de cette méthode repose sur le principe d'optimalité énoncé par le mathématicien Richard Bellman : « toute politique optimale est composée de sous politique optimale ».

1.7.1.3 Branch and Bound

Branch and Bound (La procédure par séparation et évaluation) est notée B&B, c'est une méthode le plus populaires pour résoudre des problèmes d'optimisation de manière exacte. Notamment les problèmes d'optimisation combinatoires dont on cherche à minimiser le coût de la recherche. L'utilisation de la méthode B&B nécessite :

- Une solution initiale permettant d'entamer la recherche.
- Une stratégie permettant la division du problème P en sous problèmes Pi.
- Une fonction permettant le calcul des différentes bornes.
- Une stratégie de parcours de l'arbre : parcourir en profondeur, en largeur…etc.

1.7.2 Méthodes approches

Les méthodes approchées représentent une alternative intéressante pour résoudre des problèmes d'optimisation combinatoires NP-difficiles, Les méthodes les plus connues pour leurs aptitudes à résoudre ce type de problèmes sont essentiellement les heuristiques et les métaheuristiques. Cette classe des méthodes de résolution va faire l'objet du chapitre suivant.

1.8 Conclusion

Dans ce chapitre, les définitions de base d'ordonnancement des systèmes de production ont été discuté, après les divers donnés d'un problème d'ordonnancement ont été classé, les taches, les ressources et les contraintes, dans les objectifs on a parlé sur les types de stratégie de la résolution d'un problème d'ordonnancement, ensuite défini les types d'atelier et les critères d'optimisation. Enfin nous avons sélectionné la complexité des problèmes et les différentes méthodes de résolution.

CHAPITRE 2

Les Métaheuristiques

2.1 Introduction

La résolution des problèmes d'ordonnancement d'atelier consiste à programmer un ensemble de tâches à réaliser et les affecter à un ensemble de ressources de façon à optimiser un ou plusieurs critères de performance, en respectant un ensemble de contraintes imposées, la résolution de différentes sortes de problèmes rencontrés dans notre vie quotidienne a poussé les chercheurs à proposer des méthodes de résolution et à réaliser de grands efforts pour améliorer leurs performances en termes de temps de calcul nécessaire et de la qualité de la solution proposée. Au fil des années, de nombreuses méthodes de résolution de problèmes de différentes complexités ont été proposées. Ainsi, une grande variété et des différences remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution de différents problèmes en deux classes principales.

2.2 Classification des méthodes de résolution

Les méthodes peuvent être classées en deux grandes catégories : les méthodes exactes (complètes) qui garantissent la complétude de la résolution, et les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité et en temps d'exécution. Cette classe des méthodes de résolution (les méthodes exactes) nous avons défini dans le chapitre précédent.

2.3 Méthodes approchées

Comme nous l'avons mentionné dans le chapitre précédent, les méthodes exactes nécessitent des temps de traitement prohibitifs pour résoudre les problèmes de grande taille. Pour obtenir malgré tous des solutions, des méthodes approchées ont été développées. Ces méthodes donnent des solutions certes sous-optimales, mais en un temps de calcul raisonnable**.** Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale, c'est-à-dire, trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Certains problèmes demeurent hors de portée des méthodes exactes. Les méthodes approchées, dites aussi d'approximation, constituent une alternative intéressante pour traiter les problèmes d'ordonnancement de grande taille. Elles

sont définies comme étant des procédures exploitant au mieux la structure du problème considéré afin de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible **[20]**.

2.3.1 Les heuristiques

Les heuristiques sont des critères, des principes ou des méthodes permettant de déterminer plusieurs chemins, celui qui permet d'être le plus efficace pour atteindre une bonne solution du problème considéré. Elles représentent des compromis entre deux exigences : le besoin de rendre de tels critères simples et en même temps d'établir une distinction entre les bons et les mauvais choix. La performance de ces méthodes dépend largement de la pertinence et de leur capacité d'exploiter les connaissances du problème.

Une heuristique est une règle d'estimation, une stratégie, une méthode ou astuce utilisée pour améliorer l'efficacité d'un système qui tente de découvrir les solutions des problèmes complexes [Slagle, 1971]**[21].**

2.3.2 Les Métaheuristiques

Les métaheuristiques forment une famille d'algorithmes d'optimisations visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle et la gestion de production)**[22].**

Les métaheuristiques sont généralement des algorithmes stochastiques itératifs, qui progressent vers un optimum global d'une fonction objectif. Elles se comportent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques d'un problème afin en trouver une approximation de la meilleure solution

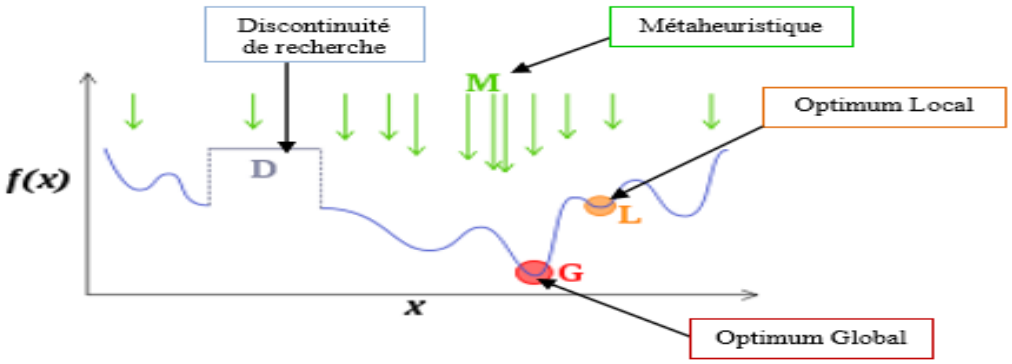


Figure 2.1 : Principe de recherche des métaheuristiques

Intérêts majeurs de cette méthode approche (métaheuristiques) est leur facilité d'utilisation dans des problèmes concrets. L'utilisateur est généralement demandeur de méthodes efficaces permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable.

Les métaheuristiques comprennent trois mécanismes : l'intensification, l'apprentissage et la diversification **[23].**

- **L'intensification** appelée aussi exploitation vise à améliorer une solution de bonne qualité en explorant les solutions trouvées.

-L'apprentissage permet à l'algorithme de tirer parti des informations contenues dans les solutions qui ont déjà été visitées.

- **La diversification** dite aussi exploration permet à la méthode d'explorer des zones de l'espace des solutions qui ont été peu ou pas visitées permettant ainsi d'éviter des optimales locaux.

L'équilibre entre la diversification et l'intensification est important.

2.3.2.1 Propriétés des métaheuristiques

Les propriétés des métaheuristiques peuvent se résumer dans quelques des points suivants [Dréo et al., 2003] **[24]** :

- Elles sont inspirées par des analogies : avec la physique (recuit simulé, …), avec la biologie (algorithmes génétiques, …) ou avec l'éthologie (colonies de fourmis, colonies d'abeilles…).
- Ce sont des stratégies au moins pour partie stochastiques.
- Les concepts de base des métaheuristiques peuvent être décrits de manière abstraite.
- Les métaheuristiques sont en général non déterministes et ne donnent aucune garantie d'optimalité.
- Les métaheuristiques peuvent faire appel à des heuristiques qui tiennent compte de la spécificité du problème traité, mais ces heuristiques sont contrôlées par une stratégie de niveau supérieur.
- Elles partagent les mêmes inconvénients : les difficultés de réglage des paramètres de la méthode et le temps de calcul élevé.

2.3.2.2 Classification des métaheuristique

Nous distinguons deux classes de métaheuristiques**[25]** :

La première classe c'est les métaheuristique à solution unique basées sur une solution et dont le principe de recherche se base sur le voisinage de la solution trouvée. Ce sont des méthodes itératives, qui à partir d'une solution initiale cherchent une amélioration possible, (algorithme de recuit simulé, algorithme de la recherche tabou, …).

Et la deuxième classe c'est les métaheuristique à population de solutionsont des algorithmes qui manipulent une population de solutions en utilisant des opérateurs. Dans cette optique, la structure générale des algorithmes évolutionnaires ou évolutionnistes enchaîne des étapes de sélection, de reproduction (ou croisement), de mutation et enfin de remplacement. Chaque étape utilise des opérateurs plus ou moins spécifiques, (algorithme génétique, algorithme colonie de fourmis, …).

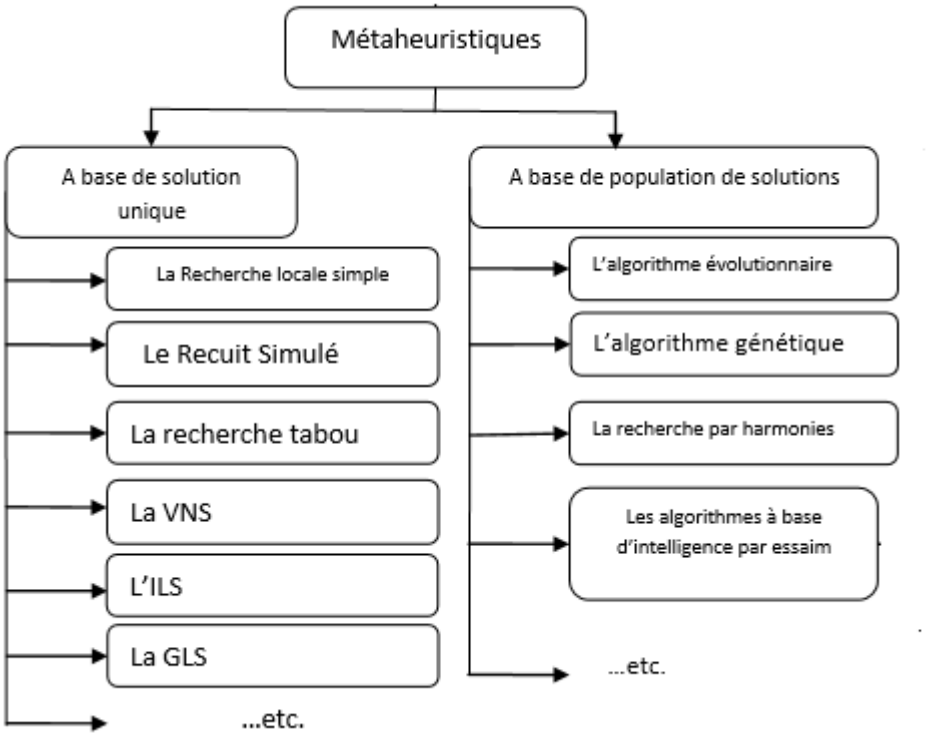


Figure 2.2 : Classification des métaheuristiques

2.3.2.2.1 Le recuit simulé

Le recuit simulé est une technique de recherche qui s'inspire du phénomène physique du recuit utilisé par les métallurgistes dont les principes ont été proposés en 1983 par des spécialistes de physique statistique (Kirkpatrick et al. [Kirkpatrick 83]) qui s'intéressaient aux configurations de basse énergie de matériaux**[26].**Le recuit simulé a eu un impact majeur sur le domaine de la recherche heuristique pour sa simplicité et son efficacité dans la résolution de problèmes d'optimisation combinatoire, bien que cette méthode ait été essentiellement développée pour la résolution des problèmes d'optimisation discrets, elle peut être utilisée pour traiter des problèmes d'optimisation continus [Talbi 2009]**[27].**

Le principe de la méthode de recuit simulé peut être résumé comme suit :

Algorithme 2.1 : Le recuit simulé

Entamer la recherche avec une solution initiale s ;
Affecter une valeur initiale à la température T ;
Calculer la fitness f(s) de la solution initiale s ;
Générer une solution s' voisine de s ;
Calculer la fitness f (s') de s' ;
Calculer l'écart de qualité (fitness) entre la solution s et la solution s' comme suit :

$$\Delta(f) = f(s') - f(s)$$

Si $\Delta(f) \geq 0$ **alors** $s \leftarrow s'$
Si non générer un nombre aléatoire $r \in [0,1]$
Si $\boldsymbol{r < exp\left(\frac{\Delta(f)}{T}\right)}$ $\boldsymbol{alors}$ $\boldsymbol{s \leftarrow s'}$

La méthode du recuit simulé, appliquée aux problèmes d'optimisation, considère une solution initiale et recherche dans son voisinage une autre solution de façon aléatoire. L'originalité de cette méthode est qu'il est possible de se diriger vers une solution voisine de moins bonne qualité avec une probabilité non nulle. Ceci permet d'échapper aux minima locaux [Jourdan, 2003]**[28]**,

Le fonctionnement de cet algorithme est illustré par la figure suivante :

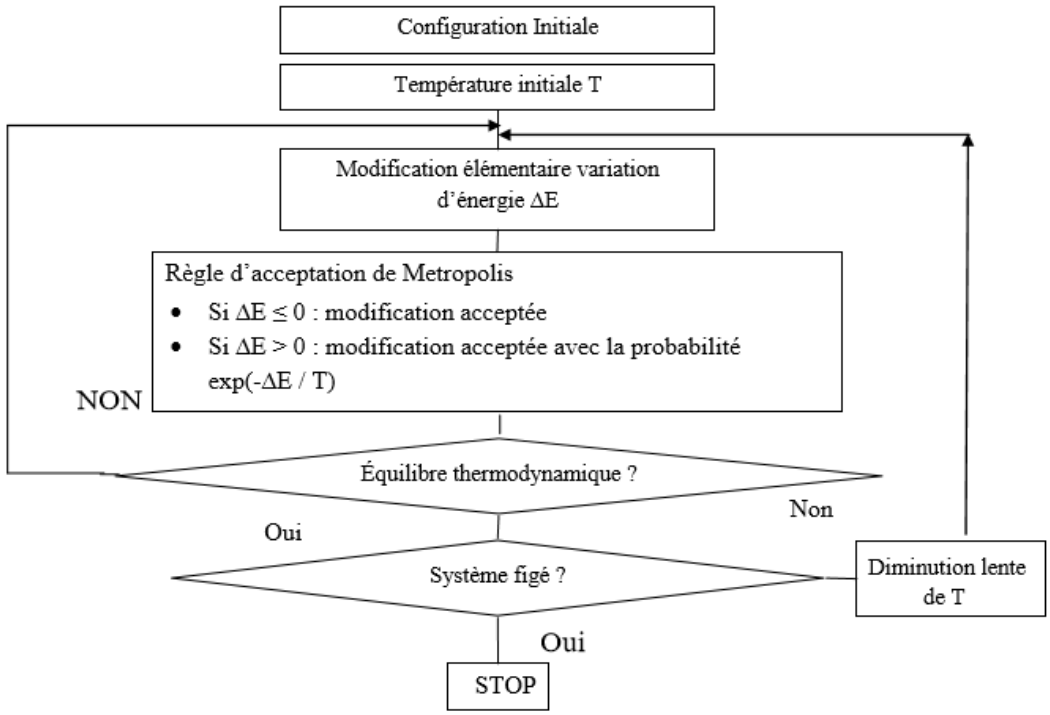


Figure 2.3 : Organigramme du recuit simulé

Le recuit simulé est un algorithme basé sur la recherche à voisinage (recherche locale). C'est un algorithme simple, facile à implémenter et à adapter à un grand nombre de problèmes : traitement d'image, sac à dos, voyageur de commerce, ordonnancement, etc**.** Cet algorithme dispose d'un nombre important de paramètres (température initiale, … etc.) pour d'ajuster. En outre, le recuit simulé est un algorithme lent surtout avec les problèmes de grande taille.

2.3.2.2.2 La recherche tabou

La recherche tabou est une métaheuristique à base d'une solution unique. Elle a été proposée en 1986 par Glover, inspirée de la mémoire humaine [Glover, 1986]. La recherche tabou est une procédure développée pour faire face à des problèmes d'optimisation discrets. Cependant, elle peut être appliquée à des problèmes continus. Contrairement au recuit simulé qui est totalement dépourvu de mémoire et qui ne génère qu'une seule solution voisine à chaque itération, cette méthode a montré une grande efficacité pour la résolution des problèmes d'optimisation difficiles**[29].**

La méthode de recherche tabou d'examine un échantillonnage de solution du voisinage de s et retient toujours la meilleure solution voisine s', même si celle-ci est de piètre qualité que la solution courante s, afin d'échapper de la vallée de l'optimum local et donner au

processus de la recherche d'autres possibilités d'exploration de l'espace de recherche afin de rencontrer l'optimum global. En fait, les solutions de mauvaise qualité peuvent avoir de bons voisinages et donc guider la recherche vers de meilleures solutions. Cependant, cette stratégie peut créer un phénomène qui on peut revisiter des solutions déjà parcourues plusieurs fois.

Le principe de la méthode de recherche tabou peut être résumé comme suit :

Algorithme 2.2 : La recherche tabou

Début

Construire une solution initiale s ;

Calculer la fitness f(s) de s ;

Initialiser une liste tabou vide ;

$s_{best} = s$;

Tant que le critère d'arrêt n'est pas vérifié **faire**

Trouver la meilleure solution s' dans le voisinage de s qui ne soit pas tabou ou qui vérifie le critère d'aspiration ;

Calculer f(s') ;

Si fitness de (s') est meilleure que fitness de (s_{best}) **alors**

s_{best} =s' ;

Fin Si

Mettre à jour la liste tabou ;

s=s' ;

Fin Tant que

Retourner s_{best} ;

Fin

La recherche tabou a été largement utilisée pour la résolution de problèmes d'optimisation difficiles comme : le problème du voyageur de commerce, les problèmes du sac à dos, les problèmes de routage d'ordonnancement, … etc. C'est une méthode facile à mettre en œuvre, rapide, donne souvent de bons résultats et permet de se sauver du premier optimum local rencontré contrairement à la méthode de la recherche locale simple, Le fonctionnement de cet algorithme est illustré par la figure suivante :

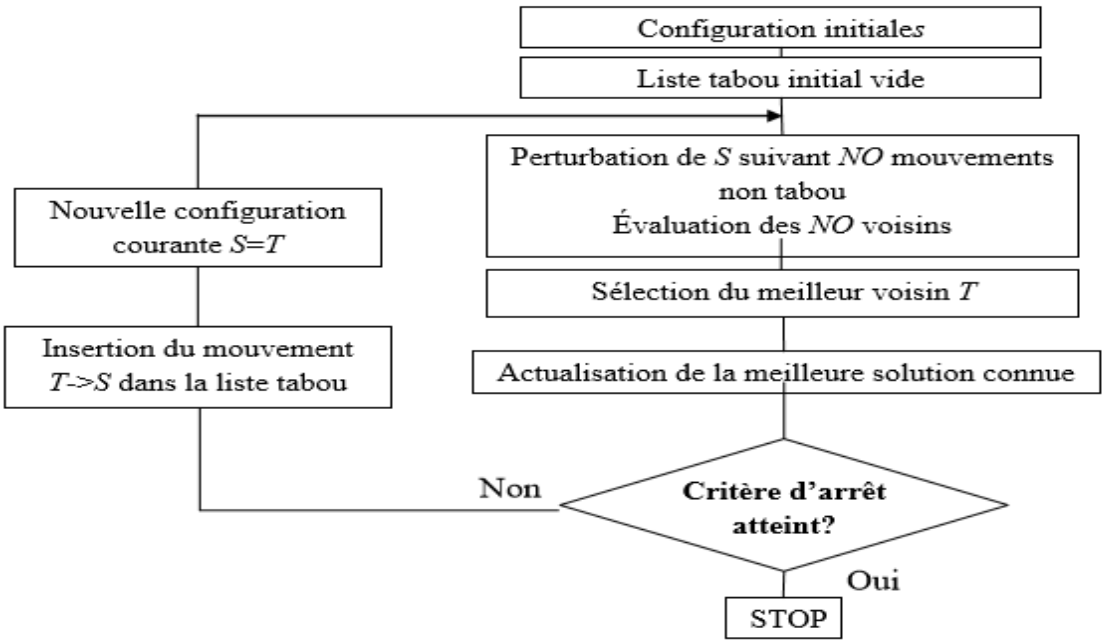


Figure 2.4 : L'organigramme de la recherche tabou

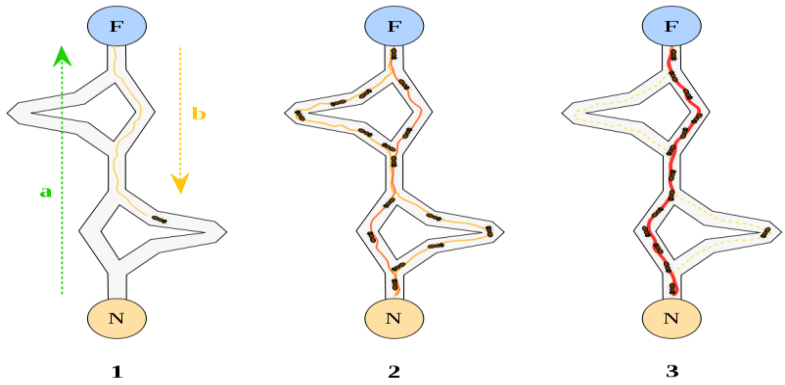
2.3.2.2.3 Les colonies de fourmis

Les algorithmes de colonies de fourmis c'est des algorithmes a été introduit par Marco Dorigo et al, au début des années 1990 [Dorigo et al, 1997]. les algorithmes de colonies de fourmis sont considérés comme des métaheuristiques à population, où chaque solution est représentée par une fourmi se déplaçant sur l'espace de recherche. Les fourmis marquent les meilleures solutions, et tiennent compte des marquages précédents pour optimiser leur recherche **[30].**

Ils sont des algorithmes inspirés du comportement des fourmis, ou d'autres espèces formant un super organisme, et qui constituent une famille de métaheuristiques destinées à résoudre les problèmes d'optimisation difficile. Ils se basent sur une technique de recherche coopérative qui imite les comportements collectifs de fourmis réelles pour la résolution de ce

genre des problèmes. L'idée originale provient de l'observation de l'exploitation des ressources alimentaires chez les fourmis. En effet, celles-ci, bien qu'ayant individuellement des capacités cognitives limitées, sont capables collectivement de trouver le chemin le plus court entre une source de nourriture et leur nid, les fourmis commencent à chercher la zone entourant leurs nids de manière aléatoire, elles peuvent construire aussi le chemin le plus court à l'aide des mécanismes de communications indirectes via une substance volatile chimique appelée phéromone, qui est le principal facteur de la coordination des activités collectives chez les fourmis.

Figure 2.5 : Expérience de sélection du plus court chemin



1) La première fourmi trouve la source de nourriture (F), via un chemin quelconque (a), puis revient au nid (N) en laissant derrière elle une piste de phéromone (b).
2) Les fourmis empruntent indifféremment les quatre chemins possibles, mais le renforcement de la piste rend plus attractif le chemin le plus court.
3) Les fourmis empruntent le chemin le plus court, les portions longues des autres chemins perdent leur piste de phéromones.

L'algorithmes de colonies de fourmis ont été conçus pour le problème du voyageur de commerce et depuis cette technique a considérablement évolué.Cette problème il se base sur trois phases essentielles [Colorni et al, 1992]**[31]** :

- La construction du trajet de chaque fourmi.
- La distribution de phéromones sur le trajet de chaque fourmi.
- Evaporation des pistes de phéromones.

Algorithme 2.3 : L'algorithme de colonies de fourmis pour le TSP

Début
Initialiser une population de m fourmis ;
Evaluer les m fourmis ;
Tant que la condition d'arrêt n'est pas satisfaite **faire**
Pour i=1 à m **faire**
Construire le trajet de la fourmi i ;
Déposer des phéromones sur le trajet de la fourmi i ;
Fin pour
Evaluer les m fourmis ;
Evaporer les pistes de phéromones ;
Fin Tant que
Retourner la ou les meilleures solutions ;
Fin

Cette algorithme représente le schéma général de l'algorithme de colonies de fourmis pour le problème du voyageur de commerce (TSP : Traveling Salesman Problem).

La procédure générale de l'algorithme de colonies de fourmis se compose de trois grandes étapes : l'initialisation de l'algorithme, la construction de la solution, et la mise à jour de la phéromone

Algorithme 2.4 : L'algorithme de colonies de fourmis
Initialiser les traces de phéromone **Répéter** **Pour** chaque fourmi **faire** Construction de solutions utilisant la phéromone Mettre à jour les traces de phéromone Évaporation Renforcement **Fin pour** **Jusqu'à** satisfaction d'un critère d'arrêt Sauvegarder la meilleure solution trouvée

Les algorithmes de colonies de fourmis ont été appliqués à un grand nombre de problèmes d'optimisation combinatoire comme : les problèmes des tournées de véhicules, le problème d'affectation quadratique Comme beaucoup de métaheuristiques, l'algorithme de base a été adapté aux problèmes dynamiques, en variables réelles, aux problèmes stochastiques, multi-objectifs ou aux implémentations parallèles, etc.

2.3.2.2.4 Optimisation par Colonie d'abeilles

L'optimisation par colonie d'abeilles est une famille récente des métaheuristiques. Son principe est basé sur le comportement des abeilles réelle dans la vie, l'algorithme a été réaliser pour la première fois vers 2004 par Craig A.Tovey à Georgia Tech, à la fin de 2004 et au début de 2005, Yang à l'université de Cambridge à développer cette algorithme pour résoudre les problèmes d'optimisation numérique [Yang, 2005].Généralement, une colonie d'abeilles contient une femelle reproductrice appelée reine, quelques centaines de mâles connus sous le nom de faux-bourdon, et de 10.000 à 80.000 femelles stériles qui s'appellent les ouvrières**[32].**

- **La Reine :** Dans une colonie d'abeilles, il y a une seule reine qui est la femelle reproductrice avec l'espérance de vie entre 3 et 5 ans. Son rôle principal est la reproduction

- **Le male (faux-bourdon)** : sont des mâles, plus gros et massifs, les yeux globuleux, présents uniquement au printemps et au début de l'été. **I**ls ne servent à rien sinon à féconder une éventuelle nouvelle reine.
- **Ouvrières :** Les ouvrières sont les abeilles femelles mais elles ne sont pas reproductrices, elles sont responsables de la défense de la ruche utilisant sa piqûre barbelée.On peut énumérer les activités des ouvrières par le critère des jours de sa vie comme suit : nettoyage de cellules, soigner les abeilles, production de la cire, surveiller les autres abeilles, et recherche de nourriture.

Le processus de recherche de nourriture chez les abeilles est fondé sur un mécanisme de déplacement très efficace. Il leur permet d'attirer l'attention d'autres abeilles de la colonie aux sources alimentaires trouvées dans le but de collecter des ressources diverses. En fait, les abeilles utilisent un ensemble de danses frétillantes comme moyen de communication entres elles. Ces danses permettent aux abeilles de partager des informations sur la direction, la distance et la quantité du nectar avec ses congénères. La collaboration et la connaissance collective des abeilles de la même colonie sont basées sur l'échange d'information sur la quantité du nectar dans la source de nourriture trouvée par les différents membres. Des études sur le comportement de danses frétillantes des abeilles ont montré par Chan et Tiwari en 2007 :

- La direction des abeilles indique la direction de la source de nourriture par rapport au soleil.
- L'intensité de la danse indique la distance de la source de nourriture.
- La durée de la danse indique la quantité du nectar dans la source de nourriture trouvée

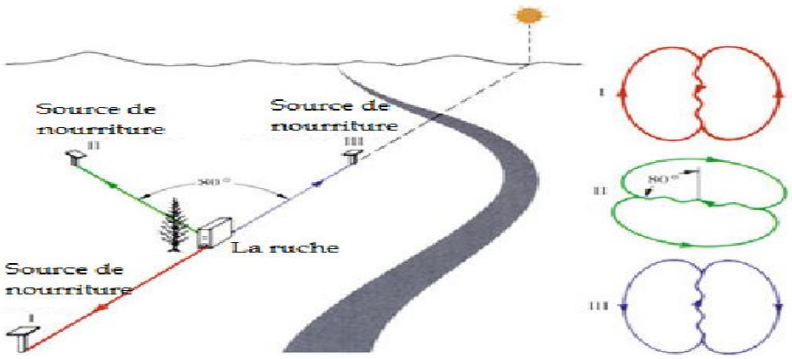


Figure 2.6 : L'indice de la direction

Figure 2.7 : la dance frétillante

L'algorithme de colonie d'abeilles, que nous proposons pour la résolution du problème de stockage de conteneurs, fait intervenir six paramètres : le nombre maximum d'itérations (NImax), le nombre d'éclaireuses parmi les abeilles (Fs), le nombre d'élites (Ne) à sélectionner parmi les sites de fleurs trouvés par les éclaireuses, le nombre (Nb) de sites à sélectionner parmi les meilleurs sites qui suivent les élites, le nombre (Fe) de butineuses à envoyer dans les sites élites, le nombre (Fb) de butineuses à envoyer dans les autres sites sélectionnés.

Algorithme 2.5 : L'algorithme de colonies d'abeilles
Chaque éclaireuse trouve une solution Évaluer chaque solution Initialiser le nombre d'itérations, i=1 **Tant que** (i < NI max) **faire** Sélectionner Ne solutions élites Sélectionner Nb solutions parmi les meilleures Fe butineuses améliore les solutions élites Évaluer les solutions améliorées, puis sauvegarder les Ne meilleures parmi elles Fb butineuses améliorent les autres solutions sélectionnées à l'étape de sélectionner Nb Évaluer les solutions améliorées, puis sauvegarder les Nb meilleures parmi elles Les abeilles restantes construisent aléatoirement de nouvelles solutions i=i+1 5. **Fin tant que**

2.3.2.2.5 Algorithme génétique

L'algorithme génétique représente une célèbre métaheuristique évolutionnaire. Il a été proposé par Jhon Holland en 1975 [Holland, 1975]. L'algorithme génétique s'inspire des mécanismes biologiques tels que les lois de Mendel et la théorie de l'évolution proposée par Charles Darwin [Darwin, 1859]. Son processus de recherche de solutions à un problème donné imite celui des êtres vivants dans leur évolution. L'algorithme génétique c'est une techniques d'optimisation itératives et stochastiques visant à résoudre des problèmes d'optimisation NP-difficiles, à utiliser le concept de populations d'individus pour faire face à des problèmes qui émergent du secteur industriel, Il utilise les vocabulaires suivants : gène, chromosome, individu, population et génération **[33]**.

- **Un gène :** est un ensemble de symboles représentant la valeur d'une variable. Dans la plupart des cas, un gène est représenté par un seul symbole (un bit, un entier, un réel ou un caractère).
- **Un chromosome :** est un ensemble de gènes, présentés dans un ordre donné de manière qui prend en considération les contraintes du problème à traiter. Par exemple, dans le problème du voyageur de commerce, la taille du chromosome est égale au nombre de villes à parcourir. Son contenu représente l'ordre de parcours de différentes villes. En outre, on doit veiller à ce qu'une ville (représentée par un nombre ou un caractère par exemple) ne doit pas figurer dans le chromosome plus qu'une seule fois.
- **Un individu :** est composé d'un ou de plusieurs chromosomes. Il représente une solution possible au problème traité.
- **Une population :** est représentée par un ensemble d'individus (i.e. l'ensemble des solutions du problème).
- **Une génération :** est une succession d'itérations composées d'un ensemble d'opérations permettant le passage d'une population à une autre.

2.3.2.2.5.1 Principe de fonctionnement de l'algorithme génétique

L'algorithme génétique fait évoluer une population composée d'un ensemble d'individus pendant un ensemble de génération jusqu'à ce qu'un critère d'arrêt soit vérifié. Le passage d'une population à une autre est réalisé grâce à des opérations d'évaluation, de sélection, de reproduction (croisement et mutation) et de remplacement.

Le processus de recherche de l'algorithme génétique est fondé sur des étape nécessaires en défini suivants :

- **Le codage des individus**

Le codage des individus permet la représentation des chromosomes représentant les individus. Cette étape associe à chaque point de l'espace de recherche une structure de données spécifique, appelée ensemble de chromosomes, qui caractérisera chaque individu de la population, le codage doit donc être adapté au problème traité

- **Génération de la population initiale**

Une étape importante lors du fonctionnement du processus de l'algorithme génétique, le choix de la population initiale peut rendre la recherche de la solution optimale du problème traité plus facile et plus rapide, la première population est générée soit d'une manière aléatoire, soit par des heuristiques ou des techniques spécifiques au problème.

- **Evaluation**

Evaluation c'est-à-dire détermination de la fonction d'adaptation (fitness) ou bien la fonction objective, généralement est doit être on mesure la performance de chaque individu

- **Sélection**

La sélection permet de favoriser la reproduction des individus qui ont les meilleures fitness (les meilleures qualités). L'opérateur de sélection doit être conçu pour donner également une chance aux mauvais éléments, car ces éléments peuvent, par croisement ou mutation.

Il existe plusieurs techniques de sélection, on définit quelques techniques :

- **Sélection aléatoire :** La probabilitéde chaque individu (P(i)) est la même (loi uniforme), doncla sélection s'effectue d'une manière aléatoire.
- **Sélection par tournoi :** Elle cela dépend de sélectionner aléatoirement plusieurs individus et ensuite on compare leurs fonctions d'adaptation et le mieux adapté est sélectionné.

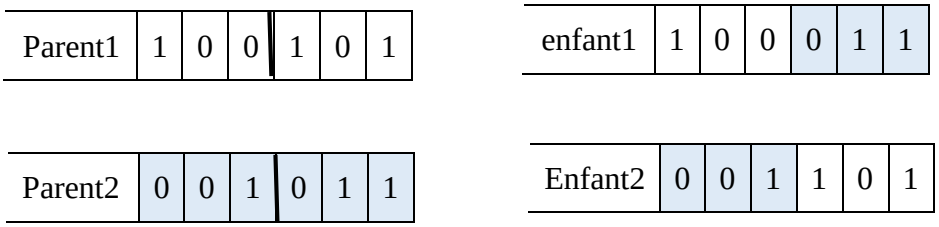
- **Croisement**

L'opérateur de croisement est un opérateur stochastique qui manipule la structure des chromosomes, le croisement fait avec deux parents et génère deux enfants, en espérant qu'un des deux enfants au moins héritera de bons gènes des deux parents et sera mieux adapté

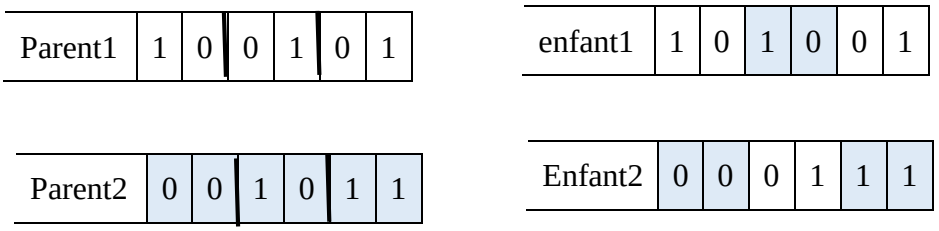
qu'eux. Le but de cet opérateur est la création de nouveaux individus en exploitant l'espace de recherche.

Il existe plusieurs méthodes de croisement par exemple le croisement en un seul point, ou croisement à deux points

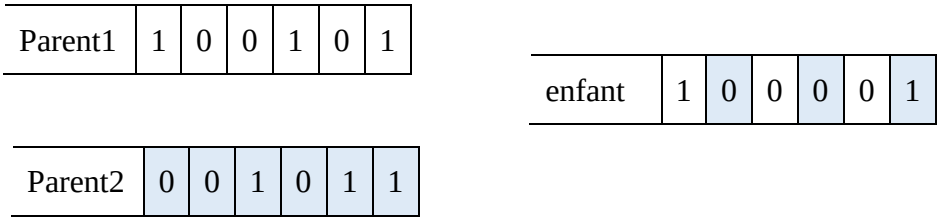
- **Croisement à un point :** Consiste à choisir aléatoirement deux parents et les diviser en deux parties à la même position (un seul point découpure). Le point de coupure s'effectue à n'importe quel niveau des gènes, donc tous les points possèdent la même probabilité d'être sélectionnés.



- **Croisement à deux points** : Consiste à choisir deux points de coupure aléatoirement pour dissocier chaque parent en trois parties.



- **Croisement uniforme :** Ce type de croisement est fondé sur la probabilité. En fait, il permet la génération d'un enfant en échangeant chaque gène des deux parents



➢ Mutation

Il consiste à modifier quelques gènes des chromosomes des individus, dans le but d'intégrer plus de diversité au sein du processus de la recherche. Dans les Algorithmes Génétiques, la mutation est considérée comme un opérateur secondaire par rapport au croisement. Parmi les stratégies de mutation utilisées en pratique :

- **La mutation uni-point :**Cette mutation se fait par altération d'une seule valeur sur le chromosome.

- **La mutation multipoints :**Cette mutation se fait par altération de plusieurs valeurs sur le chromosome.

- **La mutation par valeurs :** Cette mutation se fait par transformation d'une valeur donnée en une autre valeur déterminée, sur tous les gènes de chromosome.

➢ Le remplacement

Après croisement et mutation des individus, la phase de croisement et mutation permet de créer une nouvelle population composée de deux groupes d'individus : parents et enfants. La phase de remplacement permet de décider quels sont les individus qui vont représenter la nouvelle population.

La performance de l'algorithme génétique dépend de ses caractéristiques qui dépendent du problème. Le fonctionnement d'un algorithme génétique est résumé dans la figure suivante **[34]** :

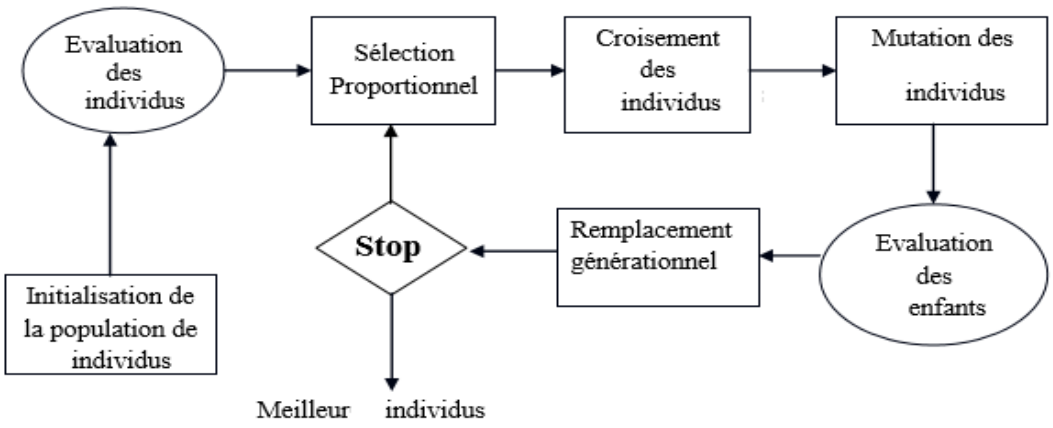


Figure 2.8 : Fonctionnement d'un algorithme génétique

L'algorithme génétique a été utilisé pour la résolution de nombreux problèmes académiques et industriels, sa simplicité et sa facilité d'hybridation avec d'autres métaheuristiques et de son application avec succès sur une très large gamme de problèmes, telle que : le problème de planification d'un système de production, la sécurité des systèmes de communication, la gestion des stocks d'une usine de production et la gestion des approvisionnements… etc.

2.3.2.2.6 Algorithme des Lucioles

Les lucioles sont de petits coléoptères ailés capables de produire une lumière clignotante froide pour une attraction mutuelle. Dans le langage courant entre les lucioles, ils sont également utilisés synonymes bogues d'éclairage ou des vers luisants. Ce sont deux coléoptères qui peuvent émettre de la lumière, mais les lucioles sont reconnues comme des espèces qui ont la capacité de voler.

Les femelles peuvent imiter les signaux lumineux des autres espèces afin d'attirer des mâles qu'elles les capturent et les dévorent. Les lucioles ont un mécanisme de type condensateur, qui se décharge lentement jusqu'à ce que certain seuil est atteint, ils libèrent l'énergie sous forme de lumière. Le phénomène se répète de façon cyclique. L'Algorithme des lucioles c'est un métaheuristique plus récente a été développé par Yang en 2008 [Yang 08] est inspiré par l'atténuation de la lumière sur la distance et l'attraction mutuelle mais il considère toutes les lucioles comme unisexes **[35].**

Figure 2.9 : Les lucioles

2.3.2.2.6.1 Paramétrages des algorithmes des Lucioles

Ces paramètres dépendent étroitement du type de problème à résoudre, le plus souvent les valeurs de ces paramètres sont réglées en fonction des résultats expérimentaux obtenus. Dans notre algorithmedes Lucioles il y a 4 paramètres importantes défini ensuivants :

- **Intensité de lumière**

Dans le cas le plus simple pour les problèmes de minimisation, la luminosité, ou l'intensité lumineuse d'une luciole à un endroit particulier x peut être choisie comme : I (x) α 1/ f (x).

- **Attractivité**

Dans l'algorithme des lucioles, la principale forme de la fonction d'attractivité peut être n'importe quelle fonction monotone décroissante telle que la forme générale suivante :

$$\beta_{i,j} = \beta_0 * e^{-\gamma r_{i,j}}$$

Où r est la distance entre deux lucioles, β0 est l'attractivité à r = 0 et γ est un coefficient constant d'absorption de lumière

- **Distance**

La distance entre 2 lucioles i et j au xi et xj peut être la distance cartésienne comme suit :$r_{i,j} =$

$$\sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2}$$

où $x_{i,k}$est la k^{eme}composante de la i^{eme} luciole

- **Mouvement**

Le déplacement d'une luciole i attirée par une plus lumineuse (attrayante) luciole j, est déterminé par :

$$x_i = (1 - \beta_{i,j})x_i + \beta_{i,j}x_j + a(rand - 1/2)$$

Où le premier terme et la second est due à l'attraction. Le troisième terme est la randomisation. α est le paramètre aléatoire et peut être constant. "rand" est un générateur de nombre aléatoire uniformément distribuée dans [0, 1].

Les principales étapes de l'algorithme Les lucioles sont données Ci-dessous :

Algorithme 2.6 : L'algorithme des lucioles

Début
Générer une population initiale de lucioles **Xi** (i=1...,n)
Déterminer les intensités de lumière Ii at **Xi** via f(**Xi**)
Tant que(t≤Nbr_iteration max) **faire**
Pour i=1 **à** n // toutes les lucioles
Pour j=1 **à** n // toutes les lucioles
Si (Ij >Ii) **alors**
Attractivité βi,j varie selon la distance ri,j
Déplacer luciole i vers j avec l'attractivité ßi,j
Sinon déplacer i aléatoirement
Fin si
Evaluer la nouvelle solution
Mettre à jour l'intensité Ii
Vérifier si luciole i est la meilleure
Fin j, Fin i
Trouver la meilleure luciole en fonction d'objective
t++
Fait
Fin

L'algorithme des lucioles permet de fournir rapidement des solutions qui sont proche de la solution optimale grâce aux mouvements des lucioles en fonction d'attractivité et d'intensité qui est défini par la fonction objectif et qui est le centre de tous les calculs, il a été développé pour résoudre les problèmes d'optimisation, mais plus tard il a été utilisé pour résoudre discrètement des problèmes tels que les vendeurs itinérants …etc.

2.4 Conclusion

Au cours de ce chapitre, nous avons présenté la classification des méthodes de résolution des problèmes combinatoires. Ensuite, nous avons cité les propriétés des métaheuristiques, les principale métaheuristiques, leurs phénomènes d‘inspiration, leurs principes de fonctionnement. Enfin, nous avons intéresse à l‘adaptation des métaheuristiques (algorithme génétique et l’algorithme de lucioles) pour adoptée dans notret

CHAPITRE 3

Adaptation de l'algorithme de luciole et l'algorithme génétique et résultats de simulation

3.1 Introduction

Dans ce chapitre, nous visons la résolution d'un problème d'ordonnancement dans un atelier de type flow shop, à l'aide de deux métaheuristiques proposées l'algorithme génétique et l'algorithme de luciole, pour faire ça nous avons présentons l'adaptation de l'algorithme de luciole et ces paramètres suivi par les résultats de simulationet la comparaison avec l'algorithme génétique.

Pour valider notre adaptation nous proposons leurs applications et simulation sur différentes classes de problèmes Flow Shop avec différentes tailles, Cette approche a été appliquée sur une population initialisée aléatoirement au départ, et qui permet par la suite de modifier itérativement cette population afin de garantir une amélioration des résultats.

Les problèmes d'ordonnancement d'ateliers de type flow-shop, ou ateliers à cheminements uniques, sont parmi les problèmes les plus connus dans le domaine de l'ordonnancement. La résolution des problèmes d'ordonnancement de type flow-shop a connu plusieurs étapes. Depuis les travaux de Johnson, l'objectif le plus visé, dans la résolution de ces problèmes d'ordonnancement est de réduire au maximum le Makespan ou Cmax, temps de fin de fabrication du dernier produit.

L'objectif de cette étude est la minimisation du temps d'exécution maximale makespan (Cmax) (Cmax =max {Cj ; j=1, ……, n}, où Cj est la date de fin d'exécution du taches) par deux Métaheuristiques, à savoir l'algorithme de lucioles et l'algorithme génétique pour avoir les meilleures solutions.

Dans le but de résoudre le problème d'ordonnancement de type flow shop proposé, nous présentons dans ce chapitre, de manière synthétique le principe et l'adaptation de l'algorithme de luciole, les résultats obtenus et la comparaison avec l'algorithme génétique.

Ce chapitre est composé de trois partie la première est ces différents paramètres de l'algorithme de luciole et la présentation de l'adaptation de cette algorithme, la deuxième partie est la présentation de l'adaptation de l'algorithme génétique et la troisième partie est la présentation du résultat de simulation de notre programme et la comparaison entre les deux algorithmes et nous terminons le chapitre par une conclusion.

3.2 Différents paramètres de l’algorithme de luciole

3.2.1 Nombre de luciole

Le nombre de lucioles aussi appelé la taille de la population initiale, à une influence directe sur l'algorithme de luciole pour cela il est très important de bien choisir ce paramètre pour garantir un meilleur compromis entre la qualité de la solution et la rapidité de l'exécution.

D’après’ les différents tests effectués, on constate que plus la taille de la population est grande, sa diversité augmente est donc la qualité de solution est meilleure.

Par conséquent si le temps d'exécution de l'algorithme augmente, il affecte l'efficacité de l'algorithme.

Par contre si le nombre de luciole est petit, il y aura alors une probabilité de converger vers un optimum local et donc il est plus efficace d'avoir un nombre important de lucioles pour assurer une diversité et éviter le problème des minimas locaux.

3.2.2 Attraction de luciole

Dans notre algorithme, il y a deux valeurs importantes : la variation de l'intensité lumineuse et la formulation de l'attractivité.

L'attraction de luciole est proportionnelle à l'intensité de la lumière vue par les lucioles adjacentes. Cette attraction est diminuée avec la distance, c.-à-d. plus la distance entre deux lucioles augmente plus l'attraction diminue et que la lumière est également absorbée par le média.

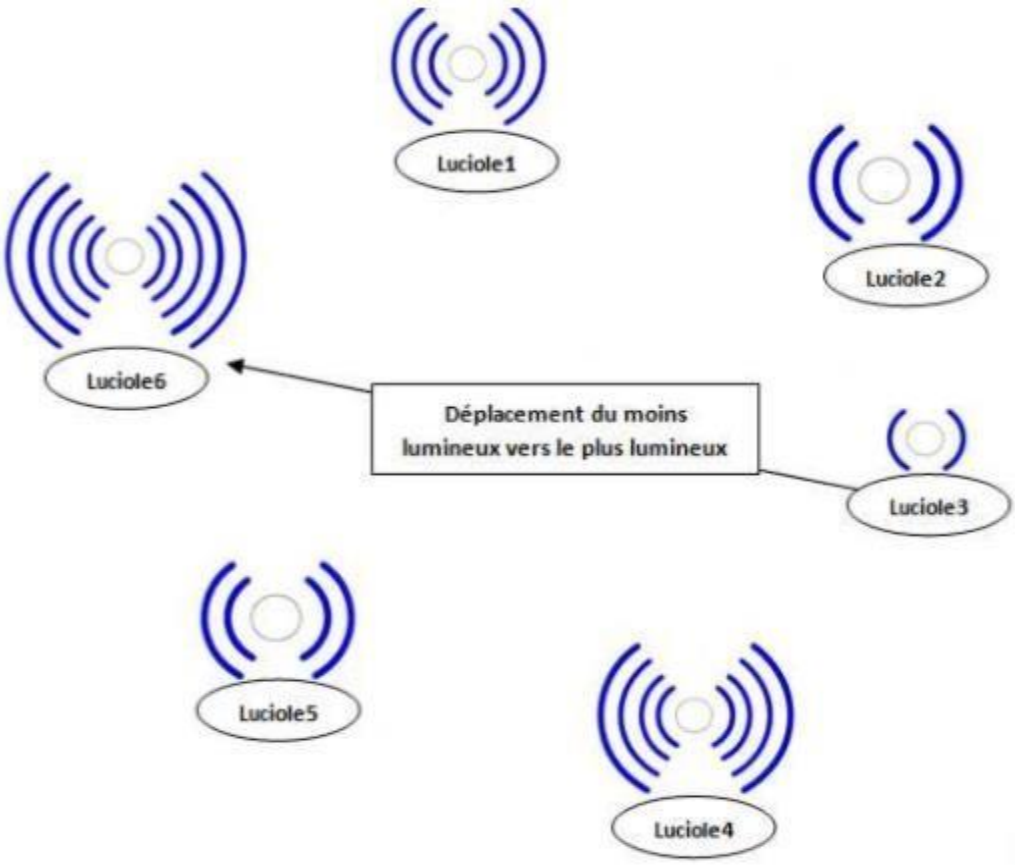


Figure 3.1 : Comment déplacer les lucioles

3.2.3 Attraction initiale

C'est l'attractivité quand la distance entre deux lucioles = 0. En général il s'agit d'un paramètre (beta0) ∈ [0,1]. Nous considérons deux valeurs limites de (beta0): (beta0)= 0 indique une recherche aléatoire non coopérative distribuée. (beta0) = 1 signifie que la recherche coopérative locale où brillante des lucioles détermine les positions des autres lucioles dans son propre quartier.

3.2.4 La distance

La distance entre deux lucioles est un paramètre très important, il est évalué de différentes manières. Pour notre algorithme, la distance entre deux lucioles est cartésienne adoptée dans un espace D-dimensionnel.

3.2.5 Coefficient d'absorption

Le coefficient d'absorption (gamma) contrôle la variation de l'attractivité en fonction de la distance entre deux lucioles communiquées. Il est dans l'intervalle [0, ∞].

gamma=0 correspond à aucun changement, pas de variation ou attractivité constante,

gamma= ∞, correspond à une recherche aléatoire complète.

Nous préférons garder la valeur de (gamma∈ [0,1]), gamma= 1 entraîne une attractivité proche de zéro qui est encore équivalente à la recherche aléatoire complète.

Ce coefficient d'absorption personnalisé pourrait être basé sur la "longueur caractéristique" de l'espace de recherche optimisé.

3.2.6 Nombre de génération

La convergence vers la solution optimale globale n'est pas garantie dans tous les cas même si les expériences dénotent la grande performance de la méthode. De ce fait, il est fortement conseillé de doter l'algorithme d'une portée de sortie en définissant un nombre maximum d'itération.

3.3 Adaptation de l'algorithme de lucioles

L'algorithme des lucioles est une métaheuristique récente bio-inspirée.sa source d'inspiration est basée sur l'émission de la lumière, absorption de la lumière et le comportement attractif mutuelle entre les lucioles. Initialement, il a été développé pour résoudre les problèmes d'optimisation. La luminosité de la lumière clignotante peut être considérée comme une fonction objective qui devra être optimisé. Les étapes de l'algorithme de luciole sont résumées comme suit :

- ✓ Créer la population initiale de lucioles (X)

Une étape importante lors du fonctionnement du processus de l'algorithme luciole, le choix de la population initiale peut rendre la recherche de la solution optimale du problème traité plus facile et plus rapide, en écrit dans le programme du Matlab

- ✓ Evaluez l'intensité de la lumière Ii de toutes les lucioles en utilisant la fonction objective (Cmax) pour chaque Xi

Evaluation c'est-à-dire détermination de la l'intensité de tous les lucioles ou bien la fonction objective (Cmax), généralement est doit être on mesure la performance de chaque luciole

- ✓ Déterminer le coefficient d'absorption (gamma)

Le coefficient d'absorption (gamma) contrôle la variation de l'attractivité en fonction de la distance entre deux lucioles communiquées (gamma∈ [0,1]), nous avons fait une étude sur l'intervalle de la coefficient d'absorption pour trouver la bonne valeur qui correspond à la simulation de l'algorithme de luciole, nous voyons les résultats de cette étude dans les pages suivantes.

- ✓ Déterminer le paramètre (beta0)

En général il s'agit d'un paramètre (beta0) ∈ [0,1], (beta0) est l'attractivité à r = 0 (r c'est la distance entre deux luciole).

- ✓ Classer les Xi (luciole)

Ou bien classer les fonction objectif (Cmax) on ordre ascendante pour calculé les distances ou bien les déférences entre les résultats

- ✓ Calculer r0=Cmax1-Cmax2

Calculé r0 (distance entre les résultats pour calculé l'intensité de l'attractivité

- ✓ Calcule l'attractivité

L'attraction de luciole est proportionnelle à l'intensité de la lumière vue par les lucioles adjacentes. Cette attraction est diminuée avec la distance, c-à-d plus la distance entre deux lucioles augmente plus l'attraction diminue et que la lumière est également absorbée par le média

Pour i=1 à nombre d'itérations

Pour j=1 à Taille de population

Pour q=1 à Taille de population

If Ij<Iq

(I) représenté l'intensité de chaque luciole j ou luciole q

Mais si l'intensité du lumière Ij>Iq, ici il y a une mutation entre j et q.

- ✓ Évaluer les intensités ou bien les Cmax

  Détermination de la fonction objective

- ✓ Classer tous les intensités (les fonction objectif)

Pour choisir la meilleure (ou les meilleurs) solution (minimiser la fonction objectif Cmax)

3.4 Adaptation de l'algorithme de génétique

L'Algorithme Génétique sont considérés par plusieurs chercheurs comme une méthode bien adaptée au problème de type Flow Shop, même si elle ne peut pas arriver à l'optimum dans certains cas difficiles.

Notre application porte uniquement sur l'Algorithme Génétique standard, c-à-d, la version de base caractérisé par une population d'individus générés aléatoirement, un opérateur de sélection, un opérateur de croisement appliqué à un point, et un opérateur de mutation. Les étapes de l'algorithme de génétique sont résumées dans l'algorithme suivant :

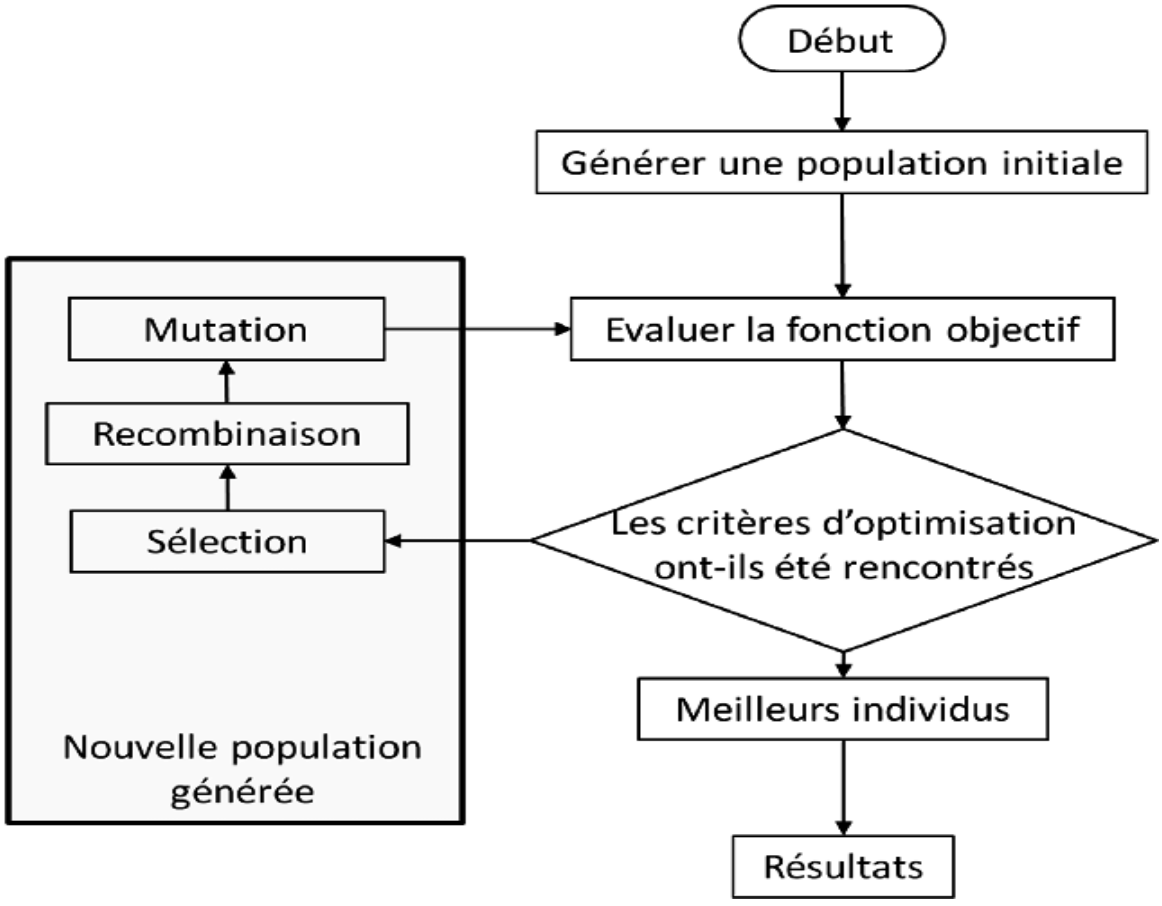


Figure 3.2 : adaptation de l'algorithmes génétique

- ✓ Génération de la population initiale

Une étape importante lors du fonctionnement du processus de l'algorithme génétique, la première population est générée soit d'une manière aléatoire, soit par des heuristiques ou des techniques spécifiques au problème.

- ✓ Evaluation de la fonction objective

Evaluation c'est-à-dire détermination de la fonction d'adaptation (fitness) ou bien la fonction objective, généralement est doit être on mesure la performance de chaque individu

- ✓ Sélection

La sélection est appliquée afin de favoriser au cours du temps les individus les mieux adaptés, à les faire se produire. Elle consiste à choisir deux individus (solutions) dits « parents » en but de créer deux autres individus dits « enfants ».

- ✓ Recombinaison ou bien Croisement

Le croisement est un opérateur très important dans les Algorithmes Génétiques. Cet opérateur est appliqué sur chaque deux parents sélectionnés

- ✓ Mutation

Il consiste à modifier quelques gènes des chromosomes des individus, dans le but d'intégrer plus de diversité au sein du processus de la recherche.

3.5 Résultats et Discussion

Dans cette section, nous réalisons plusieurs simulations sur un nombre de classes des problèmes Flow Shop.

Les simulations sont effectuées sur 5 classes de problèmes en faisant varier dans chaque classe le nombre de jobs (10jobs/20machines, 20jobs/20machines, 30jobs/20machines,

40jobs/20machines, 50jobs/20machines) et pour chaque classe nous considérons 5 exemples différents (les temps de traitement de pièces diffèrent d'un exemple à l'autre et sont pris dans un intervalle de [1 :31]).

3.4.1 L'études sur l'algorithme de luciole

L'objectif des expérimentations menées dans cette section est d'analyser l'impact de changement des paramètres de l'algorithme sur la qualité sur de ces résultats.

L'étude de sensibilité de l'algorithme de luciole dans notre travail est effectuée par rapport à sa taille de population.

- **L'étude sur la taille population**

Nous avons fait la simulation pour différentes tailles de population (Tpop=10, 20, 30, 40, 50). Trois fois pour chaque exemple, Les résultats montrent que le Cmax minimum est obtenu pour une taille de population égale à 20 (2 fois) et pour la taille de population égale à 30 (6 fois) et pour la taille de population égale à 40 (9 fois) et pour taille de population égale à 50 (8 fois).

	Tpop=10		**Tpop=20**		**Tpop=30**		**Tpop=40**		**Tpop=50**	
Exemple 1	547	523	558	523	547	553	515	**509**	516	**509**
	536		558		518		513		516	
Exemple 2	575	576	570	561	566	559	**543**	549	563	555
	567		549		556		562		560	
Exemple 3	578	571	534	540	542	**527**	565	562	554	536
	584		546		556		554		549	
Exemple 4	570	571	542	531	542	552	540	541	548	528
	580		555		549		537		**520**	
Exemple 5	592	596	575	582	574	574	606	566	**565**	575
	606		572		601		583		572	

Tableaux 3.1 : Résultats de simulation de l'algorithme de lucioles pour 5 exemples de 10 jobs 20 machines avec différente taille de population

Nous pouvons remarquer, à travers le tableau ci-dessus que le Cmax minimum est obtenu pour une taille de population égale à 40 pour les exemples 1 et 2 et pour une taille de population égale à 30 pour l'exemple 3 et pour une taille de population égale à 50 pour l'exemple 4 et 5.

	Tpop=10		**Tpop=20**		**Tpop=30**		**Tpop=40**		**Tpop=50**	
Exemple 1	855	822	847	865	837	**811**	831	813	813	853
	846		873		833		843		851	
Exemple 2	832	810	816	785	811	807	762	800	799	**699**
	816		807		813		808		813	
Exemple 3	788	816	800	795	796	795	795	788	777	778
	809		812		793		**772**		800	
Exemple 4	838	839	856	840	829	842	821	823	809	802
	857		828		**800**		824		823	
Exemple 5	838	832	832	839	824	805	**783**	836	811	816
	821		831		808		796		806	

Tableaux 3.2 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 20 jobs 20 machines avec différente taille de population

Les résultats montrent que le Cmax minimum est obtenu pour une taille de population égale à 30 pour les exemples 1 et 4 et pour une taille de population égale à 50 pour l'exemple 2 et pour une taille de population égale à 40 pour l'exemple 3 et 5.

	Tpop=10		**Tpop=20**		**Tpop=30**		**Tpop=40**		**Tpop=50**	
Exemple 1	1240	1282	1258	1269	1224	1221	**1203**	1241	1237	1261
	1248		1242		1230		1235		1237	
Exemple 2	1275	1212	1250	1209	1211	1211	1252	1229	1216	**1192**
	1246		1261		1216		1215		1203	
Exemple 3	1273	1234	1234	1292	1240	1253	1255	1250	**1208**	1271
	1280		1272		1251		1260		1245	
Exemple 4	1287	1303	1266	1274	1262	1242	1244	1301	1275	**1217**
	1309		1308		1272		1240		1274	
Exemple 5	1262	1204	1209	1213	1214	1227	1200	1199	**1135**	1222
	1230		1175		1207		1139		1203	

Tableaux 3.3 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 30 jobs 20 machines avec différente taille de population

Les résultats montrent que le Cmax minimum est obtenu pour une taille de population égale à 50 pour les exemples 1 et 2 et 3 et 4 et pour une taille de population égale à 30 pour l'exemple 5.

	Tpop=10		**Tpop=20**		**Tpop=30**		**Tpop=40**		**Tpop=50**	
Exemple 1	1240	1282	1258	1269	1224	1221	**1203**	1241	1237	1261
	1248		1242		1230		1235		1237	
Exemple 2	1275	1212	1250	1209	1211	1211	1252	1229	1216	**1192**
	1246		1261		1216		1215		1203	
Exemple 3	1273	1234	1234	1292	1240	1253	1255	1250	**1208**	1271
	1280		1272		1251		1260		1245	
Exemple 4	1287	1303	1266	1274	1262	1242	1244	1301	1275	**1217**
	1309		1308		1272		1240		1274	
Exemple 5	1262	1204	1209	1213	1214	1227	1200	1199	**1135**	1222
	1230		1175		1207		1139		1203	

Tableaux 3.4 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 40 jobs 20 machines avec différente taille de population

Les résultats montrent que le Cmax minimum est obtenu pour une taille de population égale à 40 pour les exemples 1 et pour une taille de population égale à 50 pour l'exemple 2 et 3 et 4 et 5.

	Tpop=10		**Tpop=20**		**Tpop=30**		**Tpop=40**		**Tpop=50**	
Exemple 1	1448	1413	1440	1446	1439	1422	1438	1409	**1366**	1446
	1455		1413		1419		1383		1445	
Exemple 2	1401	1437	1445	1432	1434	1436	1433	1432	1420	1433
	1467		1430		**1394**		1426		1399	
Exemple 3	1447	1439	**1379**	1417	1399	1413	1385	1409	1386	1409
	1409		1431		1430		1403		1414	
Exemple 4	1499	1509	1465	1486	1488	1469	1442	**1436**	1476	1449
	1507		1442		1475		1484		1479	
Exemple 5	1419	1426	1422	1438	**1381**	1446	1419	1393	1387	1443
	1456		1438		1397		1399		1413	

Tableaux 3.5 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 50 jobs 20 machines avec différente taille de population

Les résultats montrent que le Cmax minimum est obtenu pour une taille de population égale à 50 pour les exemples 1 et pour une taille de population égale à 30 pour l'exemple 2 et 5 et pour une taille de population égale à 20 pour l'exemple 3.et pour une taille de population égale à 40 pour l'exemple 4.

D'après ces études on remarque que les meilleurs résultats sont obtenus pour la taille population égale 40 (Tpop=40) et taille population égale 50 (Tpop =50).

- **L'étude sur le coefficient d'absorption (gamma)**

Dans cette partie nous avons fait la simulation pour différente valeur de coefficient d'absorption (gamma= (0.2,0.4,0.6,0.8,1)) et pour une taille de population Tpop=40 et pour beta0=1

	0.2		**0.4**		**0.6**		**0.8**		**1**	
Exemple 1	515	**509**	520	534	521	**509**	529	527	515	**509**
	521		515		515		521		513	
Exemple 2	561	555	556	564	587	556	554	563	**543**	549
	552		565		551		581		562	
Exemple 3	541	546	**529**	538	557	542	550	570	565	562
	555		534		561		550		554	
Exemple 4	547	555	542	553	548	540	556	540	540	541
	538		**522**		534		529		537	
Exemple 5	593	605	**559**	574	594	594	573	578	606	566
	572		575		578		575		583	

Tableaux 3.6 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 10 jobs 20 machines avec différente valeur de gamma

Les résultats obtenus, dans le tableau montre que le Cmax minimum est obtenu pour gamma=0.2 et gamma=0.6 et pour gamma=1 pour Exemple 1 et pour gamma=1 pour Exemple 2 et pour gamma=0.4 pour Exemple 3 et Exemple 4 et Exemple 5.

	0.2		**0.4**		**0.6**		**0.8**		**1**	
Exemple 1	817	854	822	858	847	833	819	819	831	**813**
	830		846		819		828		843	
Exemple 2	819	789	784	771	806	813	778	765	**762**	800
	800		802		807		785		808	
Exemple 3	792	772	766	775	782	778	**756**	781	795	788
	807		811		801		789		772	
Exemple 4	819	838	793	793	805	827	828	**774**	821	823
	799		828		814		847		824	
Exemple 5	814	790	811	788	801	808	790	812	**783**	836
	786		815		812		793		796	

Tableaux 3.7 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 20 jobs 20 machines avec différente valeur de gamma

Les résultats obtenus, dans le tableau montre que le Cmax minimum est obtenu pour gamma=1 pour Exemple 1 et Exemple 2 et Exemple 5 et pour gamma=0.8 pour Exemple 3 et Exemple 4.

	0.2		**0.4**		**0.6**		**0.8**		**1**	
Exemple 1	1012	1017	983	1055	1039	1023	**970**	1040	1024	1052
	1006		1036		1038		1064		1028	
Exemple 2	1029	1021	1040	**986**	1052	1028	1022	1022	1023	1045
	1049		1021		1026		1013		1020	
Exemple 3	1029	980	1014	1005	1002	986	1017	1022	982	992
	1028		**979**		1012		1012		1028	
Exemple 4	1028	999	1047	1009	977	983	1000	988	1017	1016
	1014		989		**965**		992		1010	
Exemple 5	1066	1057	1001	1049	**999**	1035	1061	1033	1063	1059
	1034		1011		1031		1057		1038	

Tableaux 3.8 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 30 jobs 20 machines avec différente valeur de gamma

Les résultats obtenus, dans le tableau montre que le Cmax minimum est obtenu pour gamma=0.8 dans Exemple 1 et pour gamma=0.4 dans Exemple 2 et Exemple 3 et Exemple 5 et pour gamma=0.6 dans Exemple 4.

	0.2		**0.4**		**0.6**		**0.8**		**1**	
Exemple 1	1264	1258	1237	1237	1221	1216	**1197**	1219	1203	1241
	1251		1261		1204		1200		1235	
Exemple 2	1214	1260	1226	1209	1233	**1199**	1250	1226	1252	1229
	1248		1214		1249		1237		1215	
Exemple 3	1271	1253	1246	1237	**1212**	1278	1280	1258	1255	1250
	1231		1217		1258		1264		1260	
Exemple 4	1278	1266	1241	1276	**1226**	1276	1277	1265	1244	1301
	1250		1296		1272		1266		1240	
Exemple 5	1182	1183	1201	1198	1191	1187	1204	1205	1200	1199
	1195		1225		1220		1229		**1139**	

Tableaux 3.9 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 40 jobs 20 machines avec différente valeur de gamma.

Les résultats obtenus, dans le tableau montre que le Cmax minimum est obtenu pour gamma=0.8 dans Exemple 1 et pour gamma=0.6 dans Exemple 2 et Exemple 3 et Exemple 4 et pour gamma=1 dans Exemple 5.

	0.2		0.4		0.6		0.8		1	
Exemple 1	1383	1401	1422	1422	1386	1358	1416	1448	1438	1409
	1392		1446		**1331**		1432		1383	
Exemple 2	1420	1449	1445	1413	1427	1435	1449	1449	1433	1432
	1455		**1392**		1424		1455		1426	
Exemple 3	1391	1380	1404	1403	1384	1421	1409	1414	1385	1409
	1409		**1351**		1423		1413		1403	
Exemple 4	**1392**	1482	1449	1468	1450	1432	1481	1471	1442	1436
	1431		1488		1496		1492		1484	
Exemple 5	1404	1443	1412	1391	1441	1426	1387	1409	1419	1393
	1420		1400		**1365**		1391		1399	

Tableaux 3.10 : résultats de simulation de l'algorithme de lucioles pour 5 exemples de 50 jobs 20 machines avec différente valeur de gamma.

Les résultats obtenus, dans le tableau montre que le Cmax minimum est obtenu pour gamma=0.6 dans Exemple 1 et Exemple 5 et pour gamma=0.4 dans Exemple 2 et Exemple 3 et pour gamma=0.2 dans Exemple 4.

D'après ces études on remarque que les meilleurs résultats sont obtenus pour valeur de coefficient d'absorption égale 0.6 (gamma=0.6) et valeur de coefficient d'absorption égale 1 (gamma=1).

3.4.2 Comparaison entre les deux techniques :

Pour une comparaison entre les deux algorithmes, nous avons pris les mêmes paramètres de simulation suivants : taille de population (Tpop=40) et valeur de coefficient d'absorption(gamma=1) pour l'algorithme de luciole.

	10jobs×20mach		**20jobs×20mach**		**30jobs×20mach**		**40jobs×20mach**		**50jobs×20mach**	
	AG	**Luciole**	**AG**	**Luciole**	**AG**	**Luciole**	**AG**	**Luciole**	**AG**	**Luciole**
Exemple 1	522	**509**	836	**813**	1034	**1024**	1224	**1203**	1419	**1409**
Exemple 2	550	**543**	782	**762**	1023	**1020**	1231	**1215**	1429	**1426**
Exemple 3	**554**	**554**	786	**772**	1004	**982**	1261	**1250**	1409	**1385**
Exemple 4	539	**537**	826	**821**	1013	**1010**	1249	**1240**	1462	**1436**
Exemple 5	570	**566**	796	**783**	1040	**1038**	1175	**1139**	1405	**1393**

Tableau 3.11: Comparaison des résultats de simulation entre les deux techniques

Le tableau représente les résultats obtenus par la simulation de l’AG et ceux de l’algorithme de luciole pour les différents exemples des différentes classes. Pour la taille de population (Tpop=40). Les expérimentations que nous avons effectuées ont bien montré l'efficacité l'algorithme de Luciole.

Il est clairement remarquable que l’algorithme de luciole est plus performant que l’AG en terme de qualité du résultat pour toutes les classes des problèmes, mais le temps de simulation est plus long dans les grandes tailles de population.

Comme perspectives, il serait intéressant d’étudier l'impact des paramètres de (beta0) sur l'algorithme.

3.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l‘adaptation de l’algorithme de luciole pour la simulation sur différent classes des problèmes pour étudier l'impact des paramètres sur les performances de l‘algorithme de luciole pour résoudre le problème de minimisation de makespan dans un atelier de type flow shop. Les résultats obtenus montrent que l’algorithme de luciole donne de meilleurs résultats que L’AG.

CONCLUSION GENERALE

Ce mémoire se situe dans le cadre d'adaptation et la simulation des métaheuristiques récentes pour la résolution d'un problème d'ordonnancement dans un atelier de type flow shop.

Pour la validation de notre adaptation nous avons appliqué l'algorithmes de luciole et l'algorithme génétique sur différentes classes de problèmes de différentes tailles composées de plusieurs exemples. Les résultats de simulation ont montré que l'algorithme de luciole dépasse l'algorithme génétique en termes de qualité des solutions obtenues.

Nous proposons comme perspectives de ce travail d'hybrider l'algorithme de luciole avec une autre métaheuristique de recherche locale, des méthodes exactes, ou d'autres métaheuistiques peut augmenter leur efficacité de fonction objectif

Ou bien de changer la fonction objective de makespan (Cmax) vers le retard (L).

Bibliographies

[1]: F.A. Rodammer et K. Preston White, « A recent survey of production scheduling ». IEEE Transaction on Systems, Man and Cybernetics, 1999.

[2] : J. Carlier et P. Chrétienne, « Problèmes d'ordonnancement, Modélisation, Complexité, Algorithmes ». Edition Masson, Paris, 1988.

[3]P.Esquirol et P.Lopez,, L'ordonnancement, Série: Production et techniques quantitatives appliquées à la gestion, collection Gestion, Economica, Paris, 1999.

[4]: Parunak, H.V.D., Manufacturing experience with the contract net. In Proceedings of the Fith Workshop on Distributed Artificial Intelligence, 1985.

[5] : P.Esquirol et P.Lopez,, L'ordonnancement, Economica, Paris, 1999.

[6] : Philipe Babtiste, Emmanuel Néron, Francis Sourd Modèles et algorithmes en ordonnancement, Ellipses, Paris, 2004.

[7] : P. Lopez et Roubellat, Ordonnancement de la production, Hermes Science Europe Ltd, 2001

[8] : C. Gagné, L'ordonnancement industriel : stratégies de résolution métaheuristique et objectifs multiples, Thèse PhD, 2001.

[9] : B. Grabot, Ordonnancement d'ateliers manufacturiers, Techniques de l'ingénieur Doc AG 3 105

[10]: B. Montreuil, Fractal layout organization for job shop, int. j. prod. res, 1999

[11] : Hentous H., contribution au pilotage des systèmes de production de type Job Shop, Thèse de Doctorat, INSA Lyon, 1999.

[12] : Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford stein, introduction to algorithme, MIT Press, 2009

[13] : Sylvain Perifel, Complexite algorithmique, Ellipses, 2014

[14] : Garey et johnson 1979, Section 3.1 ET Probleme SP1 dans Appendix A.3.1.

[15]: M. SAKAROVICH, Optimisation combinatoire, HERMANN, Paris, France, 1984

[16]: Sanjeev Arora et Boaz Barak, Computational Complexity: A Modern Approach, Cambridge Univercity Press, 2009.

[17]: M.R. Garey and D.S. johnson. Computers and Intractability: A Guide to the theory of NP-Completeness. New Nework: w. H. Freeman. 1983.

[18]:I. Charon, A. Germa et O. Hudry. Méthodes d'optimisation combinatoire. Masson, Paris. 1996.

[19] : DRISS IMEN**,** Analyse D'un Système Job Shop AspecOrdonnancemetn, Thèse de Doctorat, Batna, 2016.

[20] : Boumediene merouanehocine, les problemes d'ordonnancement a machines parallèles, de taches dépendantes. Magister. Blida, juillet 2006.

[21] J. R. Slagle. Artificial intelligence: The heuristic programming approach. McGraw-Hill.pp. 3. New York, 1971

[22]: Yang, X. S. (2010). Nature-inspired metaheuristic algorithms. Luniver Press

[23]: Blum, C. Roli, A. (2003) Meta-heuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM ComputSurv 35 :268–308.

[24]: Dréo, J. Pétrowski, A. Siarry, P. and Thaillard, E. (2003). Métaheuristiques pour l'optimisation difficile. Eyrolles.

[25] : Duvivier, D., Etude de l'hybridation des métaheuristiques, application à un problème d'ordonnancement de type jobshop, Thèse de doctorat, Laboratoire d'Informatique du Littoral, Cote-d'Opale, Calais. (2000).

[26]: Kirkpatrick, S. Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, 220(4598), 671-680.

[27]: Talbi, E. (2009). Metaheuristics: From Design to Implementation. JohnWiley and Sons, Inc.

[28] : Jourdan, L. (2003). Métaheuristiques pour l'extraction de connaissances : Application à la génomique. Thèse de doctorat, Université de Lille, France

[29]: Glover, F. (1986). Future paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, 13(5), 533-549.

[30]: Dorigo, M. and Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. BioSystems, 43(2), 73-81

[31]:A. Colorni, M. Dorigo, V. Maniezzo. Distributed Optimization by Ant Colonies. Proceedings of the 1rst European Conference on Artificial Life. pp 134-142, Elsevier Publishing.

[32] : H. Boukef, F. Tangour et M. Benrejeb. Sur la formulation d'un problème d'ordonnancement de type flow-shop d'ateliers de production en industries pharmaceutique. Journées Tunisiennes d'Electrotechnique et d'Automatique, Hammamet, 2006.

[33] : Holland, J. H. (1975). Adaptation in Natural and ArtificialSystems. University of Michigan Press, Ann Arbor, MI, 1975

[34]: Dréo, J. Pétrowski, A. Siarry, P. and Thaillard, E. (2003). Métaheuristiques pour l'optimisation difficile. Eyrolles.

[35] : Yang, X. S. (2008). Nature-InspiredMetaheuristicAlgorithms. First ed. Luniver Press, Frome